JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN
ALLG. BWL UND WIRTSCHAFTSINFORMATIK
UNIV.-PROF. DR. AXEL C. SCHWICKERT

Reader zur Vorlesung im Hauptstudium

Modellierung von IuK-Systemen

Wintersemester 07/08

Univ.-Prof. Dr. Axel C. Schwickert

Wirtschafts-
informatik

Grundstudium

Theoretische Grundlagen des Software Engineering

Dr. Alexander Teubner, Münster

Software Engineering stellt Methoden und Werkzeuge für die Entwicklung, den Betrieb und die Wartung von Software bereit. Als wissenschaftliche Grundlage dient die Systemtheorie. Das Systemdenken kommt im Rahmen von Requirements Engineering, Information Systems Engineering und Information Engineering auch bei der Entwicklung von Informationssystemen zum Tragen.

1. Historische Entwicklung

Begriffliche Abgrenzung

Der Begriff „Software Engineering" wurde bereits 1968 geprägt, als sich eine Konferenz mit diesem Titel in Garmisch-Partenkirchen den Problemen zuwandte, welche die Entwicklung großer Programmsysteme — und daraus resultierend ein notwendigerweise arbeitsteiliger Entwicklungsprozess — mit sich brachte. Die Begriffswahl (**Engineering**: engl. für Ingenieurwissenschaft, Ingenieurwesen) sollte zum Ausdruck bringen, dass Softwareentwicklung, wie die Entwicklung anderer komplexe technische Produkte auch, nach wissenschaftlichen Prinzipien und Vorgehensweisen erfolgen muss. Die Entwicklung und Wartung von Software sollte ein systematischer, disziplinierter und quantifizierbarer Arbeitsprozess sein. Der Begriff **Software Engineering** wurde damit bewusst gegen die damals vertretene Anschauung gesetzt, nach der die Programmentwicklung als geistig-kreative Tätigkeit im Sinne einer Kunst zu verstehen sei.

Dadurch, dass die Softwareentwicklung als technische Konstruktionsaufgabe begriffen wird, erschließt sie sich einer (ingenieur-)wissenschaftlichen Betrachtung. Wissenschaftliche Grundlage sowohl für die Erforschung und Entwicklung von Methoden, Verfahren und Werkzeugen der Softwareentwicklung im Software Engineering als auch für deren Anwendung ist die Systemtheorie.

Frage 1: Gegen welche Auffassung von Softwareentwicklung ist der Begriff „Software Engineering" gesetzt? Welchen Ansatz zur Entwicklung komplexer Softwareprodukte betont er?

2. Systemtheorie als Grundlage

Die wissenschaftliche Grundlage für die Methoden- und Werkzeugentwicklung bietet die Systemtheorie. Im Folgenden wird eine kurze Einführung in das Systemdenken gegeben. Darauf aufbauend wird die Bedeutung des Systemdenkens für die Ingenieurwissenschaften aufgezeigt und speziell die Anwendung auf die Softwareentwicklung verdeutlicht.

2.1. Grundzüge der Systemtheorie

Systemtheorie und fachwissenschaftliche Theorie

Die **Systemtheorie** stellt eine einheitliche Terminologie und Denkweise sowie Modelle und Methoden zur Beschreibung und Erklärung von Sachverhalten zur Verfügung. Sie bietet einen allgemeinen, abstrakten Bezugsrahmen, losgelöst von einem realen Inhalt. Dadurch unterscheidet sich die Systemtheorie von der fachwissenschaftlichen Theorie, die unmittelbar in den Kategorien des Erkenntnisbereichs formuliert ist. Im Unterschied zur fachwissenschaftlichen Theorie ist die Systemtheorie eine Theorie der Theoriebildung. Sie wird hier deshalb als Metatheorie bezeichnet; ihre Disziplin ist die Wissenschaftstheorie.

General Systems Theory

Den Durchbruch in der modernen Wissenschaft erreichte die Systemtheorie durch die Arbeiten des Biologen L. van Bertalanffy, der versuchte, allgemeingültige Aussagen über das Verhalten von Systemen zu formulieren. Ausgangspunkt für seine Betrachtungen waren biologische und natürliche physikalische Systeme, deren Systemeigenschaften er auf soziale Systeme übertrug. Als Folge gewann in den Jahren nach 1945 die **Idee einer allgemeinen Systemtheorie** (General Systems Theory) an Gestalt, welche die Gemeinsamkeiten physikalischer, biologischer und gesellschaftlicher Systeme aufdecken sollte. Aus Sicht der allgemeinen Systemtheorie stellt sich die Wirklichkeit als Netzwerk biologischer, psychologischer, gesellschaftlicher, ökonomischer, ökologischer und sonstiger Phänomene dar.

Die Sichtweise der Wirklichkeit als Zusammenwirken von Systemen wurde schon frühzeitig in der **Kybernetik** adaptiert, die technische, biologische und gesellschaftliche Systeme unter Lenkungsaspekten untersucht. Während in der allgemeinen Systemtheorie die Erörterung statisch-struktureller und abstrakt-theoretischer Aspekte von Systemen im Vordergrund steht, befasst sich die Kybernetik mit dynamisch-funktionalen Aspekten konkreter Systeme. Bei der allgemeinen Systemtheorie einerseits und der Kybernetik andererseits handelt es sich um unterschiedliche, vom gemeinsamen Systemdenken ausgehende Betrachtungsweisen der gleichen Phänomene. Beide Disziplinen werden deshalb heute oft als Teildisziplinen einer übergreifenden Systemtheorie verstanden.

2.1.1. Allgemeine Systemtheorie

Systemumwelt, Systemelemente und Beziehungen

Unter einem **System** (von griechisch: συστασις — Zusammenstellung, Zusammenordnung) wird eine Menge von Elementen verstanden, zwischen denen Beziehungen und Wechselwirkungen bestehen und die gegenüber der Umwelt abgegrenzt sind. Die definitorischen Elemente des Systembegriffs sind demnach die Elemente, die Beziehungen und die Abgrenzung des Systems zur Umwelt:

- **Elemente** sind Teile eines Systems, die nicht sinnvoll weiter aufgebrochen werden können. Die Zerlegung ist dabei eine Frage der Zweckmäßigkeit. Elemente stellen die kleinsten betrachteten Einheiten dar, deren interne Struktur bei einem gegebenen Zweck nicht weiter interessiert.
- Unter **Beziehungen** versteht man konstante Verbindungen, die zwischen den Elementen existieren. Das gesamte Beziehungsgefüge der Systemelemente definiert formal die Ordnung eines Systems. Sie wird als Systemstruktur bezeichnet.
- Unter der **Umwelt** des Systems versteht man Elemente, die außerhalb der Grenzen des betrachteten Systems liegen. Ausschlaggebend für die Abgrenzung des Systems gegenüber der Umwelt ist die Intensität der Beziehungen zwischen den Elementen. Charakteristisch ist, dass innerhalb der Systemgrenzen ein größeres (stärkeres, wichtigeres) Maß an Beziehungen besteht bzw. wahrgenommen wird als zwischen System und Umwelt.

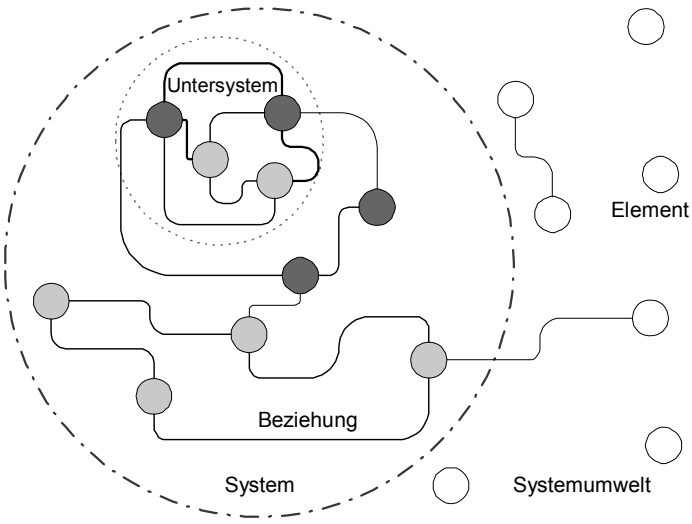


Abb. 1: Allgemeine Darstellung eines Systems

Der Aufbau eines Systems ist in Abb. 1 grafisch veranschaulicht. Die Dicke der Kanten in der Abbildung deutet die Intensität der Beziehungen zwischen den Elementen an, die gestrichelten Kreise markieren die Grenzen des Systems.

Gliederung von Systemen: Unter- und Teilsysteme

Jedes betrachtete System kann und wird in der Regel Bestandteil eines übergeordneten Systems sein. Dieses wird als **Übersystem** bezeichnet. Gleichzeitig kann ein System, gegebenenfalls über mehrere Stufen hinweg, in untergeordnete Systeme, so genannte **Untersysteme**, aufgeteilt werden. Untersysteme sind in Bezug auf die ihnen übergeordnete Ganzheit Teile und in Bezug auf ihre Teile Ganzheiten. Jedes System lässt sich zudem unter ganz bestimmten Aspekten untersuchen; es wird dann gewissermaßen durch einen bestimmten Filter betrachtet, der ausgewählte Teile des Systems in den Vordergrund treten lässt. Die Elemente und Beziehungen, die unter einem bestimmten Aspekt zusammengefasst werden, heißen **Teilsysteme**. Sowohl Untersysteme als auch Teilsysteme werden häufig auch als **Subsysteme** bezeichnet.

Die Frage, ob bestimmte Beziehungen und Elemente ein Unter- oder Teilsystem eines übergeordneten Systems sind, lässt sich nicht generell beantworten. Sie hängt davon ab, wie die hierarchische Struktur des Bezugssystems definiert ist. Empirisch können Unter- und Teilsysteme abgegrenzt werden, indem die Zugehörigkeit der Elemente untersucht wird: Ein Element eines Untersystems kann nicht gleichzeitig Element eines anderen Untersystems sein, bezogen auf dasselbe Übersystem. Hingegen kann ein Element eines Teilsystems ohne weiteres auch Element eines anderen Teilsystems sein.

In Abb. 1 sind vier Elemente zu einem Untersystem zusammengefasst und durch eine (Unter-)Systemgrenze gegenüber den anderen Elementen des Systems abgegrenzt. Zwei der Elemente des Untersystems sind gleichzeitig Gegenstand einer Teilsystembetrachtung. Die Elemente des Teilsystems sind in der Abbildung dunkelgrau unterlegt.

Frage 2: Wie lässt sich ein System identifizieren und gegenüber der Umwelt abgrenzen?

Arten von Systemen: Offenheit und Dynamik

Hinsichtlich ihrer Aktivität werden statische und dynamische Systeme unterschieden. Während in einem **statischen System** die Beziehungsinhalte und das Verhalten von Elementen unveränderlich sind, ändern sich diese in dynamischen Systemen. In letzteren können die Elemente mehrere Eigenschaften annehmen und über wechselnde Inhalte der Beziehungen, über die sie mit anderen Elementen verbunden sind, auch deren Eigenschaften beeinflussen. Durch die Angabe der möglichen Realisationen aller betrachteten Eigenschaften der Elemente eines Systems ist der Raum der potenziellen **Systemzustände** definiert. Der ständige Übergang eines Systems von einem Zustand in den anderen macht das **Systemverhalten** aus.

Systeme sind grundsätzlich durch eine Systemgrenze von der Umwelt abgegrenzt. Ist diese Systemgrenze undurchlässig für Energie-, Materie- und Informationsflüsse, so wird das System als **geschlossenes System** bezeichnet. Demgegenüber pflegen **offene Systeme** einen Austausch mit ihrer Umwelt. Änderungen des Zustands und damit das Systemverhalten offener Systeme hängen nicht nur von internen Transformationsprozessen ab, sondern auch von den Austauschbeziehungen mit der Umwelt.

Offene Systeme sind in der Regel dynamisch, vollständig geschlossene Systeme dagegen statisch. Der Zweck offener Systeme ist es, einen von der Umwelt erhaltenen Input in irgendeiner Form zu verarbeiten und das Ergebnis als Output an die Umgebung abzugeben. Geschieht dies nicht, d.h., ist ein System offen und statisch, so ist das System „tot“. Der Input verlässt das System unverändert als Output.

Frage 3: Welcher Zusammenhang besteht zwischen dem Systemverhalten und der Offenheit bzw. Geschlossenheit eines Systems?

Struktur von Systemen: Gebilde- und Prozess-Struktur

Im Zusammenhang mit der Unterscheidung statischer und dynamischer Systeme ist eine Differenzierung des Strukturbegriffs notwendig. Während statische Systeme allein durch die Gebilde-Struktur definiert werden können, muss für dynamische Systeme zusätzlich die Prozess-Struktur beschrieben werden:

- Mit **Gebilde-Struktur** wird der statische Aspekt des Strukturbegriffs bezeichnet. Die Gebilde-Struktur kennzeichnet die feste Anordnung der Elemente eines Systems.
- Mit der **Prozess-Struktur** werden Regelungen zur Dynamik eines Systems angesprochen. Die Prozess-Struktur ordnet die Abläufe von Prozessen innerhalb der Gebildestruktur.

Offene, dynamische Systeme führen Operationen aus, die Wirkungen auf die Umwelt haben. Gleichzeitig nehmen sie Inputs aus der Umwelt auf, die das gezeigte Verhalten aus-

lösen. Systeme, die in dieser zweifachen Weise mit der Umwelt verbunden sind, werden speziell in der **Kybernetik** betrachtet.

2.1.2. Systemtheorie und Modellbildung

Modellbegriff

Die Systemtheorie liefert neben einem terminologischen Instrumentarium Prinzipien für die Modellbildung, mit denen sich die konstituierenden Aspekte des abstrakten Gegenstands „System" erfassen lassen. Unter einem **Modell** wird allgemein eine Beschreibung oder ein Bild von etwas, dem Original oder Urbild, verstanden. Modelle repräsentieren ein Original immer für die Zwecke eines erkennenden und/oder handelnden Subjekts. Ein Modell ist demnach ein (Ab-)Bild eines Originals **für einen Verwender bezüglich eines Zwecks**. Für die Untersuchung von Systemen werden vorwiegend semantische Modelle eingesetzt. Semantische Modelle erfassen ein Urbild in der besonderen Weise von Sprache, d.h., zur Darstellung werden Zeichen mit einer vereinbarten Bedeutung verwendet.

Frage 4: Was versteht man unter einem „Modell"?

Prinzipien für die Systemmodellierung

Die Systemtheorie bietet drei Prinzipien für die Modellierung von Systemen an:

- hierarchische Betrachtung
- funktionale Betrachtung
- strukturale Betrachtung

In Abb. 2 ist die Wirkung dieser drei Prinzipien auf die Modellbildung (in der Folge von links nach rechts) visualisiert. Es wird deutlich, dass die Anwendung der Prinzipien jeweils zu unterschiedlichen (semantischen) Modellen eines Systems führt.

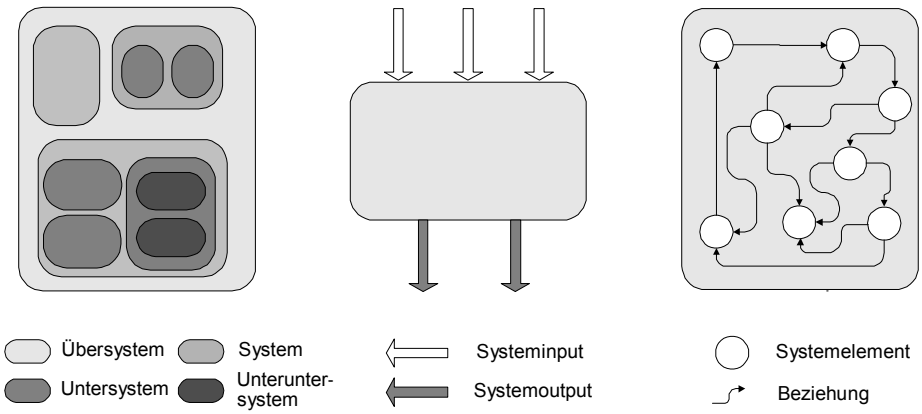


Abb. 2: Visualisierung des Systemdenkens

Die **hierarchische Betrachtung** untersucht die System-Untersystem-Zusammenhänge eines Systems. Durch das Konstrukt des „Untersystems" ist es möglich, die Teile eines übergeordneten Systems zunächst auf grober und abstrakter Ebene darzustellen. Untersysteme können dann wiederum als eigenständige Systeme begriffen und weiter verfeinert werden, bis schließlich die Ebene von Elementen erreicht ist. Durch die schrittweise Auflösung eines Gesamtsystems ergibt sich dann eine gegebenenfalls mehrfach geschachtelte Hierarchie von Untersystemen. Geht man die Hierarchie abwärts, so erhält man detailliertere Erklärungen des Systems; geht man aufwärts, so gewinnt man ein tieferes Verständnis seiner Bedeutung. Die hierarchische Betrachtung führt zu einer stufenweisen Auflösung eines Systems auf unterschiedlichen Ebenen der **Abstraktion**.

In der **funktionalen Betrachtung** werden die Funktionen bzw. Leistungen eines Systems untersucht. Dabei wird ein Systemzweck unterstellt, der durch Transformation eines Inputs in einen Output erreicht wird. Jedes System erhält Leistungen aus der Systemumwelt. Dieser Input wird von dem System in eine Leistung transformiert, die es als Output wiederum an die Umwelt oder an andere Systeme liefert. In der funktionalen Betrachtung interessiert nur, was das System leistet. Dies lässt sich aus der Differenz zwischen Input und Output feststellen. Das System selber kann deshalb als **Black Box** betrachtet werden. Die funktionale Betrachtung fokussiert die **Aufgaben** eines Systems in seiner Umwelt bzw. dessen Wirkung auf diese.

In der **strukturalen Betrachtung** verschiebt sich der Fokus von der Frage, was das System leistet, zu der Frage, wie diese Leistungen erbracht werden. Die strukturale ergänzt die funktionale Betrachtung, indem sie die Black Box öffnet und die Mechanismen be-

trachtet, durch welche die Leistungen eines Systems zustande kommen. Aus der Black Box der funktionalen Betrachtung wird in der strukturalen Betrachtung eine **White Box**. Sie zeigt die Elemente eines Systems und die Beziehungszusammenhänge, die zwischen diesen bestehen. Die White-Box-Betrachtung zeigt die **Funktionsmechanismen** des Systems, welche die zuvor in der Black-Box-Betrachtung durch die Input-Output-Beziehungen beschriebenen Leistungen des Systems bewerkstelligen.

Durch das Prinzip der hierarchischen Betrachtung wird nicht vorgegeben, nach welchem Kriterium ein System aufgelöst wird. Grundsätzlich kommen als Auflösungskriterien sowohl die Struktur als auch die Funktion in Frage. Je nach Anwendung kommt es dann zur Bildung struktureller oder funktionaler Untersysteme. **Strukturale Untersysteme** sind Cluster von Systemelementen, deren innerer Beziehungszusammenhang intensiver ist als die Beziehungen zu Elementen außerhalb des Untersystems. Sie ergeben sich auf der Grundlage einer Analyse der Beziehungen. **Funktionale Untersysteme** sind dagegen abstrakte Gesamtheiten, die nur auf rein analytischem Wege anhand der Aufgabe abgegrenzt werden können.

Frage 5: Welche Eigenschaften von Systemen werden jeweils in funktionalen, strukturalen und hierarchischen Modellen herausgestellt?

3. Anwendung der Systemtheorie auf die Gestaltung komplexer technischer Produkte

Systemtheorie in der wissenschaftlichen Forschung und technischen Gestaltungspraxis

Die Systemtheorie wurde ursprünglich entwickelt, um biologische, physikalische und sozio-ökonomische Systeme besser beschreiben und verstehen zu können. Die Anwendung der Systemtheorie zu wissenschaftlichen Zwecken wird als **Systemforschung (Systems Research)** bezeichnet. Die Denkmodelle, Grundprinzipien und Vorgehensweisen der Systemtheorie lassen sich ebenso auf die Gestaltung künstlicher Gegenstände (d.h. von Menschen geschaffenen Gegenständen, im Unterschied zur Natur) anwenden. Wird das Systemdenken auf Probleme der Gestaltung von Systemen angewendet, so wird von **Systemtechnik (Systems Engineering)** gesprochen.

Die Systemtechnik fand zuerst in den klassischen Ingenieurdisziplinen Anwendung, die sich mit der Gestaltung physikalisch-technischer Systeme befassen (Maschinenbau, Elektrotechnik usw.). Sie lässt sich jedoch ebenso auf die Gestaltung von Informationssystemen als sozio-technischen Systemen anwenden, denn wie bei der Gestaltung physikalisch-technischer Systeme steht auch dort ein Denken in Wirkungszusammenhängen im Vordergrund. Das Systemdenken ist heute die methodische Leitlinie nicht nur für das Software Engineering, sondern für alle Disziplinen der Informationssystementwicklung.

3.1. Systemtechnik in den klassischen Ingenieurdisziplinen

Das Systemdenken kennzeichnet heute nahezu alle Bereiche innerhalb der Ingenieurdisziplinen. Es zeigt sich in Aufgabenbezeichnungen wie beispielsweise Systemplanung, Systemanalyse und Systementwicklung. Diese Begriffsprägungen weisen darauf hin, dass das **Objekt der Gestaltung** als System begriffen wird. Darüber hinaus wird mit dem Begriff des Engineering die Forderung nach systematischen Vorgehensweisen zur Problemlösung verbunden. Dies ist ein Hinweis darauf, dass auch der **Gestaltungsprozess** als System betrachtet werden kann. Die Systemtechnik lässt sich demnach zum einen auf den Entwicklungsprozess **(Prozess-Sicht)** und zum anderen auf das Ergebnis des Prozesses **(Ergebnis-Sicht)** anwenden.

Systemtechnik in der Prozessgestaltung

Betrachtet man den **Prozess** der Planung und Entwicklung von (technischen) Gegenständen als System, so kann er durch Anwendung des Prinzips der hierarchischen Systemauflösung nach sachlogischen Kriterien in funktionale Subsysteme zerlegt werden. Jedes funktionale Subsystem fasst einen Komplex von Aufgaben innerhalb des Entwicklungsprozesses zusammen. Dieser wird üblicherweise als Phase bezeichnet. Zwischen den Phasen bestehen Beziehungen: Phasen erbringen Leistungen, die sie an andere Phasen weitergeben. Zudem stehen die Phasen in einer ablauflogischen und meist auch in einer temporalen Ordnung.

In der Regel wird der Entwicklungsprozess nicht nur grob in Phasen zerlegt, sondern die Phasen werden weitergehend bis auf die Ebene einzelner Problemlösungsschritte detailliert. Die Problemlösungsschritte bilden Vorschriften für die Bewältigung der Aufgabenstellung innerhalb der Phasen ab. Die Zerlegung in Phasen und Problemlösungsschritte

führt zu einem (System-)Modell des Entwicklungsprozesses, wie es Abb. 3 zeigt. Die Phasen sind als Ovale dargestellt, die Problemlösungsschritte innerhalb der Phasen als Kreise. Für die einzelnen Phasen sind beispielhaft die Leistungsbeziehungen (breite, vertikale Pfeile), für die Problemlösungsschritte zeitlich-ablauflogische Beziehungen (schmale Pfeile) angegeben. Das in Abb. 3 gezeigte Modell wird als **Prozessmodell** bezeichnet.

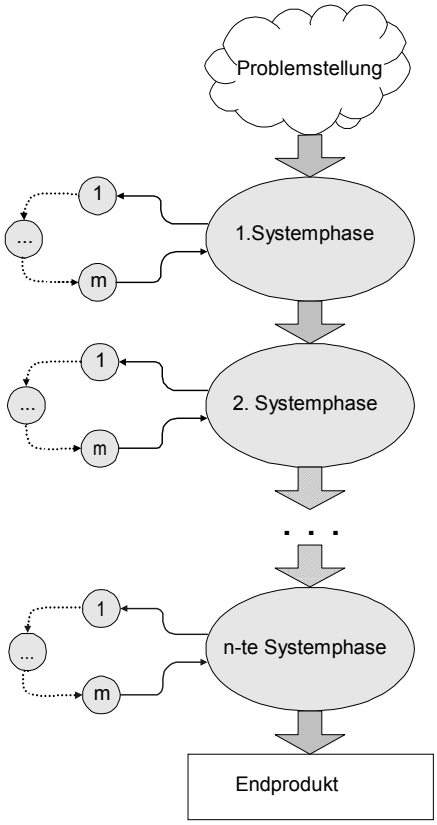


Abb. 3: Systemtechnisches Prozessmodell

Systemtechnik in der Produktgestaltung

Im Rahmen des Entwicklungsprozesses bzw. -projekts werden Ergebnisse produziert, die sich ebenfalls als Systemmodelle beschreiben lassen. Die Anwendung der Systemtechnik auf das End- oder die Zwischenergebnisse der Systementwicklung führt zu so genannten **Ergebnismodellen**. Aus einer hierarchischen Betrachtung heraus lässt sich die Zerlegung des zu gestaltenden Systems in seine Komponenten modellieren. Die funktionale Betrachtung eignet sich zur Beschreibung der Leistungen, die das System oder dessen Komponenten erbringen sollen. Mit Hilfe struktureller Modelle lassen sich die Funktionsmechanismen beschreiben, durch die diese Leistungen sichergestellt werden.

Abb. 4 zeigt eine Beschreibung der Komponenten (hierarchische Sicht), eine Beschreibung der Aufgaben bzw. Leistungen (funktionale Sicht) und eine Beschreibung der Funktions- bzw. Konstruktionsmechanismen (strukturale Sicht) als Systemmodell (vgl. Abschnitt 2.1.1).

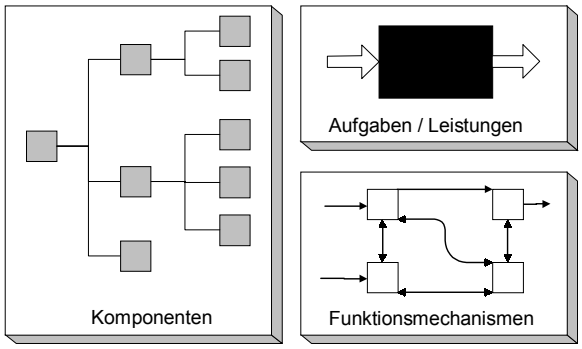


Abb. 4: Arten systemtechnischer Ergebnismodelle

Frage 6: Wodurch unterscheiden sich Prozess- und Produktmodelle?

3.2. Systemtechnik im Software Engineering

Adaption der Systemtechnik im Software Engineering: Prozessmodelle und Produktmodelle

Auch im Software Engineering wird die Systemtechnik sowohl hinsichtlich des Entwicklungsprozesses als auch bezüglich der im Prozess erzielten Ergebnisse angewendet. Der linke Teil von Abb. 3 zeigt ein Prozessmodell, das die Softwareentwicklung in fünf Phasen (Ovale) einteilt. Jede Phase von der Problemanalyse bis zur Implementierung stellt eine komplexe Teilaufgabe im Entwicklungsprozess dar. Jede Teilaufgabe transformiert einen bestimmten Input in einen Output. Der Output jeder Phase ist ein Systemmodell, das ein (Zwischen-)Ergebnis im Entwicklungsprozess dokumentiert. Ausgehend von der Problemstellung wird zuerst ein funktionales Systemmodell (Black Box) erstellt, das die Leistungsanforderungen an das zu entwickelnde Softwaresystem beschreibt. Diese Anforderungsspezifikation wird dann durch hierarchische Dekomposition schrittweise in strukturale Modelle (White Box) aufgelöst. In Abhängigkeit von der Verfeinerungsebene stellen diese Dokumente die grobe Architektur, die algorithmischen Strukturen und schließlich die maschineninterpretierbaren Anweisungen und Kontrollstrukturen dar, d.h. den Programmcode.

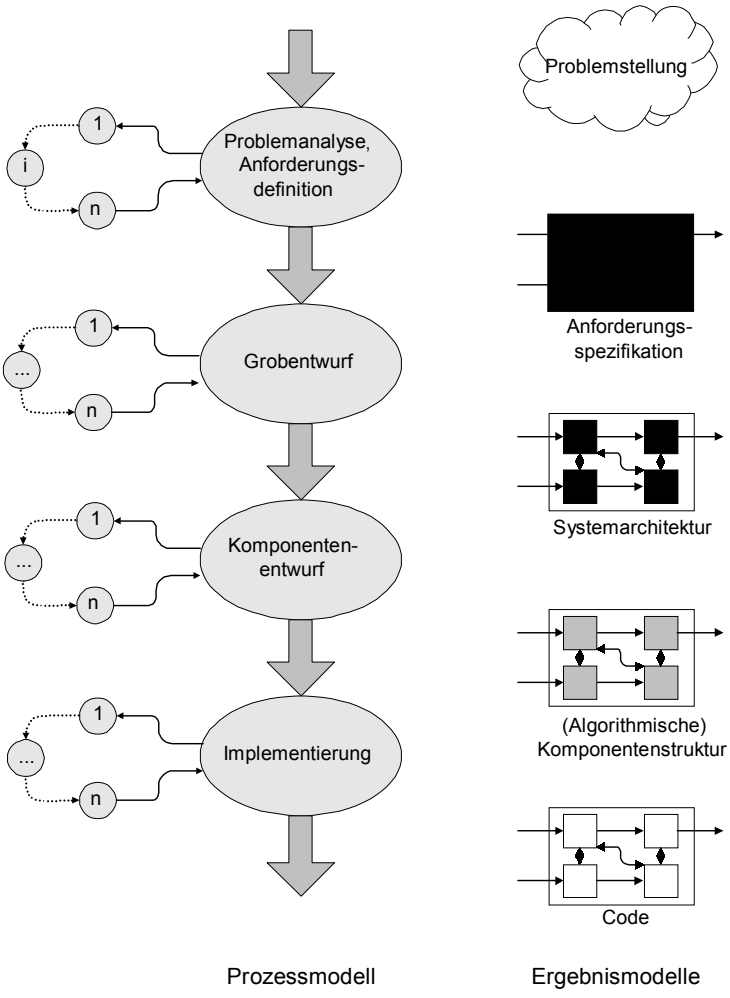


Abb. 5: Systemtechnik in der Softwareentwicklung

Prozess- und Ergebnissicht

Zwischen Prozess- und Ergebnissicht besteht ein enger Zusammenhang: In der Ergebnissicht wird unter Anwendung des Prinzips der hierarchischen Systemauflösung eine Black-Box-Betrachtung des Softwaresystems sukzessive in White-Box-Betrachtungen aller Subsysteme des Softwaresystems aufgelöst. In der Prozess-Sicht führt das Prinzip der hierarchischen Systemauflösung zu einem Vorgehen nach dem Prinzip der schrittweisen Verfeinerung. Jede Phase vollzieht einen Schritt in der Auflösung und Konkretisierung der Ergebnisse. Deshalb werden parallel zu der Aufgabengliederung in Phasen die Zwischenergebnisse der Phasen erstellt (Analysephase — Anforderungsspezifikation, Grobentwurfsphase — Softwarearchitektur usw.). Die Phasen-Zwischenergebnisse

stellen einen Reifezustand der Ergebnisse dar, der Grundlage für die Aufgaben der nächsten Phase ist. In Abb. 5 sind die Ergebnisse der unterschiedlichen Auflösungsstufen jeweils auf der Höhe der korrespondierenden Leistungsbeziehungen (breite Pfeile) zwischen den Phasen angeordnet.

Frage 7: Erläutern Sie die Unterschiede von Prozess- und Produktmodellen im Software Engineering!

4. Zusammenfassung

Mit „Software Engineering“ wird die Ingenieur-Disziplin bezeichnet, die sich mit der Entwicklung von Software beschäftigt. Charakteristisch für die Ingenieurwissenschaften und damit auch für das Software Engineering ist die Anwendung des Systemdenkens. Sowohl das Produkt „Software“ als auch der Entwicklungsprozess werden im Software Engineering als System verstanden. Die Softwareentwicklung stellt sich demzufolge als systematischer Prozess der fortlaufenden Bildung und Konkretisierung von Systemmodellen dar (System Engineering). Ausgangspunkt ist ein fachliches Problem, das in ein Anforderungsmodell übersetzt und weitergehend auf unterschiedliche Abstraktionsstufen und mit fortschreitendem Konkretisierungsgrad in eine Software-Lösung überführt wird.

Literaturempfehlungen:

Balzert, H.: Lehrbuch der Software-Technik. Heidelberg/Berlin/Oxford 1997.
Daenzer, W.F./Huber, F. (Hrsg.): Systems Engineering — Methodik und Praxis. 9. Aufl., Zürich 1997.
Heinrich, L.J.: Systemplanung. Band 1. 7. Aufl., München/Wien 1996 . Band 2. 5. Aufl., München/Wien1994.
Hesse, W./Merbeth, G./Frölich, R.: Software Entwicklung: Vorgehensmodelle, Projektführung, Produktverwaltung. München/Wien 1992.
Jantsch, E./Seiffert, H.: System, Systemtheorie. In: Seiffert, H./Radnitzky, G. (Hrsg.): Handlexikon zur Wissenschaftstheorie. 2. Aufl., München 1994, S. 329 - 338.
Pomberger, G./Blaschek, G.: Software Engineering. Prototyping und objektorientierte Software-Entwicklung. 2. Aufl., München/Wien 1996.
Stachowiak, H.: Modell. In: Seiffert, H./Radnitzky, G. (Hrsg.): Handlexikon zur Wissenschaftstheorie. 2. Aufl., München 1994, S. 219 - 222.
Teubner, R.A.: Organisations- und Informationssystemgestaltung. Theoretische Grundlagen und integrierte Methoden. Wiesbaden 1999.
Wallmüller, E.: Software-Qualitätsmanagement in der Praxis. 2. Aufl., München/Wien 2001.

Die Beantwortung der Fragen erfolgt im WISU-Repetitorium.

Lösungen des WISU-Check up von Seite 667:

1 a,b,e,f 2 b 3 a,b,d 4 a 5 a 6 a,b,c 7 c 8 e 9 — 10 — 11 a,b 12 a,b 13 c 14 — 15 a,b 16 b 17 b 18 a,b 19 c 20 a,b,c 21 a,b 22 a 23 b,c 24 d

Über sieben Brücken musst Du geh'n

Eine Kritik am Software-Engineering

Jens Coldewey

Noch bis vor wenigen Jahren galt das Software-Engineering als goldener Weg, um die Probleme der Softwareentwicklung in den Griff zu bekommen. Bei allen guten Ideen und Ansätzen dieser Richtung mehren sich aber doch die Stimmen, die das ingenieurmäßige Vorgehen in Frage stellen. Für Berater stellt sich hier schnell die Gretchenfrage. In dem Artikel werden die wichtigsten Kritikpunkte vorgestellt und Alternativen angerissen.

Früher war die Programmierung von Computern eine Kunst. Virtuos jonglierten die Damen und Herren in ihren weißen Mänteln mit Registern, Lochkarten und dem immer viel zu kleinen Speicher, um beachtliche Leistungen zu erbringen. Höhepunkt dieser Techniken war wohl die Mondlandung, die mit Ressourcen gemeistert wurde, mit denen heute kaum mehr ein Maustreiber zurecht käme. Damals wurden Programme üblicherweise in Assembler geschrieben; die Programmierer mussten die Maschine genau kennen und jede Möglichkeit wurde genutzt, um Speicherplatz oder Rechenzeit zu sparen. Die Systeme dienten anfangs meist mathematischen Aufgaben, später kamen einfache Datenbanken und Steuerungssysteme hinzu.

In den 60er Jahre wurden dann höhere Programmiersprachen modern. Statt der Maschine mühsam jeden einzelnen Schritt vorzuschreiben, wurden Sprachen entwickelt, die vom Menschen leichter verstanden wurden. Sprachen wie ALGOL-60 und später ALGOL-68 und FORTRAN deckten erfolgreich mathematische Aufgabenstellungen ab, COBOL setzte sich bald für kaufmännische Anwendungen durch. Die Beherrschung einer bestimmten Programmiersprache wurde zu einer Schlüsselqualifikation in der DV-Branche. Schon bei der Entwicklung von COBOL wurde damals ein Anspruch erhoben, der bis heute nicht wirklich eingelöst wurde: Die Programmierung der Computer sollte den Fachbereichen selbst möglich sein, die Abhängigkeit von den Programmierern sollte endlich vorbei sein. Seitdem dieser Anspruch das erste Mal formuliert wurde, hat sich das Programmieren – oder die Softwareentwicklung, wie es heute heißt – zu einem der tragenden Zweige der westlichen Wirtschaften gemausert; einschlägige Firmen wie Microsoft, Oracle oder SAP sind Schwergewichte in den jeweiligen Börsenindizes. Alteingesessene Wirtschaftsnationen zittern vor der Kompetenz von Schwellenländern und weltweit verdienen viele Millionen Frauen und Männer ihren Lebensunterhalt mit Programmieren. Längst haben die Sozialwissenschaftler die nächste industrielle Revolution ausgerufen, die Abhängigkeit von der programmierenden Zunft ist nicht beseitigt, sondern mittlerweile total.

Die Systeme wurden immer leistungsfähiger – und immer komplexer. Bald wuchs die Software ihren Erstellern über die Köpfe, die Entwicklungsmannschaften wurden immer größer und die Fehlerraten immer eindrucksvoller. Bald wurde die „Softwarekrise" erklärt – es musste etwas passieren! Mehr und mehr wurde klar, dass die Entwicklung der Software nicht nur virtuoses Programmieren möglichst ausgefeilter Algorithmen umfasst, sondern die Beherrschung der Komplexität eine immer wichtigere Rolle spielt. Zwei der wichtigsten Impulse dazu kamen von Männern, die unterschiedlicher kaum sein können: Edsger Dijkstra und Alan Kay.

Edsger Dijkstra, meines Wissens ursprünglich Numeriker, war nicht nur federführend an der Entwicklung der strukturierten Programmierung beteiligt, sondern entwickelte auch die „Abstrakten Datentypen", aus denen sich die Prinzipien der Modularisierung und Datenkapselung ableiten. Die Idee ist einfach: Man zerlege das komplizierte System in kleine, übersichtliche Teile, die kein Wissen über gegenseitige Interna besitzen und über genau definierte Schnittstellen miteinander kommunizieren. Damit war das wichtigste technische Hilfsmittel geschaffen, um die immer größer werdenden Systeme der 70er und 80er Jahre noch im Griff zu behalten.

Alan Kay, damals noch ein junger Forscher an den Xerox Laboratorien, ging einen anderen Weg. Er verglich Software mit den komplexesten Systemen, die uns bekannt sind: mit Lebewesen. Alle höheren Lebewesen bilden sich aus Zellen. Jede Zelle hat einen bestimmten Zelltyp und ist auf eine bestimmte Aufgabe spezialisiert. Von jedem Zelltyp gibt es in einem Körper abertausende Exemplare, die zwar weitgehend gleich aufgebaut sind, aber doch ihre eigene Identität besitzen. Die Zelle ist von der Außenwelt durch eine Zellmembran geschützt und kommuniziert mit dieser Umwelt über genau definierte Kanäle, ohne diese aber genauer zu kennen. Auch wenn Zellen über ihre gesamte Lebenszeit ihren Typ beibehalten, so sind sie doch in der Lage, neue Zellen zu erzeugen, die leicht angepasste Eigenschaften besitzen. In Ermangelung einer besseren Metapher nannte Kay ein entsprechendes Pendant in der Software „Objekt" und schuf damit die Basis für die objektorientierte Programmierung.

Diese beiden Ideen zeigen bei flüchtiger Betrachtung einige deutliche Parallelen. Insbesondere die Idee des geheimen Inneren, das nur über definierte Schnittstellen mit dem Außen kommuniziert, scheint beiden Ansätzen gemeinsam. Nicht übersehen darf man dabei aber, dass beide Ansätze völlig unterschiedlichen Philosophien folgen: Dijkstras Ansatz ist in bester westlicher Wissenschaftstradition analytisch: Ein gegebenes, großes System wird solange in kleinere Teil-

Jens Coldewey (E-Mail: jens_coldewey@acm.org) ist unabhängiger Berater. Er hilft großen Organisationen bei der Einführung und Umsetzung objektorientierter Konzepte.

systeme zerlegt, bis die einzelnen Bestandteile beherrschbar sind – Cäsars Prinzip „teile und herrsche" findet sich hier wieder. Kay hingegen schwebt eher die Synthese vor: Aus vielen kleinen Einheiten setzt sich ein großes System zusammen. Idealerweise existieren diese Einheiten bereits zuvor und können sich selbst an ihre neue Umgebung anpassen. Es ist müßig zu erwähnen, dass der Großteil der Softwareprojekte heute eher dem analytischen Ansatz folgt, während Kays Synthese noch immer visionär erscheint, auch wenn so manche Veröffentlichung über Komponenten gerne den gegenteiligen Eindruck erweckt. Mit verantwortlich dafür ist wohl auch der Umstand, dass bis heute kaum eine gängige Programmiersprache die Ideen Kays auch nur annäherungsweise umsetzt. Am ehesten kann meines Erachtens Smalltalk noch ein Gefühl dafür vermitteln, wohin ein solcher Ansatz führen könnte.

Die Wurzeln des Software-Engineering

Neben den Verbesserungen der Programmiertechniken hielt noch ein ganz anderer Ansatz Einzug in die Softwareentwicklung: die Idee, Software ingenieurmäßig zu entwickeln. Warum sollte man Software nicht entwickeln können, wie man etwa ein Chemiewerk baut? Hier werden zunächst die Anforderungen gesammelt, dann werden Pläne von zunehmender Granularität erstellt, bis schließlich die Bauaufträge an die einzelnen Spezialfirmen erteilt werden, die dann die funktionierende Fabrik bauen. Konsequent umgesetzt, sollte es möglich sein, Software mit der gleichen Zuverlässigkeit und Kostentreue herstellen zu können, wie heute Häuser und Fabriken gebaut werden. Wie geplagte Bauherren vermutlich wissen, verfällt nach deutschem Standesrecht zum Beispiel die Honorarforderung eines Architekten, wenn die Baukosten den Voranschlag um mehr als 30% überschreiten. Eine solche Regelung würde – übertragen auf die Softwareindustrie – sicherlich zum Ruin der halben Branche führen.

Die Verheißung pünktlicher und kostentreuer Softwarelieferung mit Hilfe von Techniken, die man sich aus der Bauindustrie abschaut, hat einen deutlichen Charme. Dem bis vor wenigen Jahren noch üblichen künstlerischen Chaos bei der Entwicklung von Software wird ein definierter Entwicklungsprozess entgegen gestellt, der sicherstellt, dass das System auch pünktlich ausgeliefert wird – zumindest in der Theorie. Prozesse, die dieser Philosophie folgen, definieren eine Reihe von Dokumenten – auch als *Artefakte* bezeichnet – die vor der eigentlichen Programmierung zu erstellen sind. Da gibt es Anforderungsdokumente, Geschäftsprozessanalysen, Grobspezifikation, Feinspezifikation, Oberflächenspezifikation, Testspezifikation, Architektur, Grobdesign, Feindesign, Moduldesign, Schnittstellenentwürfe, Testdesign, Oberflächendesign usw. In großen Projekten sind diese Dokumente nicht nur für das eigentliche System zu verfassen, sondern auch für die Entwicklungsumgebung, für die Testumgebung, für das Konfigurationsmanagement, für die Einführung und für die Wartung. Was hier fast wie Kabarett klingt, ist lediglich eine komprimierte Sicht auf die heute gängigen Entwicklungsprozesse, wie z. B. das V-Modell (vgl. [Brö95]) oder den „Rational Unified Process" (vgl. [Jac99]). Wegen der vielen zu erstellenden Unterlagen werden solche Prozesse als „schwergewichtige Prozesse" bezeichnet.*

* Von den meisten schwergewichtigen Prozessen existieren mittlerweile iterative Varianten, die zumindest schnellere Auslieferungen und bessere Priorisierungen versprechen. An dem Problem, dass durch die vielen Dokumente bzw. Artefakte die Änderbarkeit erschwert wird und hohe Aufwände in die Erstellung dieser Artefakte wandern, ändern diese Vorschläge aber nichts.

Theorie und Praxis

Meine persönliche Erfahrung mit Kundenprojekten, aber auch als Projektleiter sieht da leider etwas anders aus. Projekte, die mit schwergewichtigen Prozessen arbeiten, zeigen immer wieder einige typische Symptome:

- *Die Dokumente sind schlecht geschrieben.* Wichtige Details werden in ihnen übersehen, Festlegungen sind unpräzise. Klassisch ist zum Beispiel die Anforderung, ein System müsse dialogfähig sein. Daraus werden dann von Auftragnehmer und -geber meist unterschiedliche Schlussfolgerungen gezogen, wie schnell das System nun auf bestimmte Anfragen zu antworten hat – eine Quelle langen und unergiebigen Streits. Aber auch Architekturbeschreibungen, die eher Verkaufsprospekten zu 3-*Tier*-Architekturen ähneln als dem hinterher umgesetzten System, sind keine Seltenheit.
- *Die Projekte geraten in Verzug.* Gerade wenn engagierte Projektleiter versuchen, möglichst gute Anforderungs- und Analysedokumente zu erstellen, geraten sie bereits in den frühen Projektphasen in Verzug. Werden die Anforderungen anfangs nicht genau genug festgehalten, geraten sie spätestens während Analyse und Design in Zeitprobleme. Häufig wird versucht den Verzug durch massive Überstunden wieder einzuholen. Es scheint mittlerweile als normal angesehen zu werden, dass zumindest Projektleiter eine 80-Stunden-Woche fahren. Dabei sollte man nicht vergessen, dass jemand, der 80 Stunden pro Woche arbeitet, keinen sozialen oder psychischen Ausgleich mehr zu seinem Beruf hat. Ihm oder ihr fehlen also alle Puffer zur Bewältigung von Stress und schwierigen Situationen. Unter solchen Bedingungen auch noch seine soziale Kompetenz zu erhalten, grenzt an eine Aufgabe von herkulischem Ausmaß.
- *Im gesamten Team werden massiv Überstunden geleistet.* Bisher habe ich allen meinen Kunden, die unter diesem Symptom litten, Ed Yourdons Meisterwerk „Death March" empfohlen, das die wohl beste Analyse dieses Phänomens enthält, die ich bisher gelesen habe (siehe [You97]).
- *Gerade in objektorientierten Projekten sind Design- und Codequalität häufig katastrophal.* Von Wiederverwendbarkeit kann keine Rede sein, die Systeme haben erhebliche Stabilitäts- und Performanzprobleme.
- *Die Auslieferung erfolgt meist zu spät und dann auch nicht in vollem Umfang.* Oft schließen sich lange Streitereien zwischen Auftragnehmer und Auftraggeber über die Einschätzung von Fehlern und über kostenlose Nachbesserungen an. Für keinen der Beteiligten ist das eine besonders erquickliche Situation.

Ich vermute, dass es kaum einen Leser gibt, dem diese Symptome nicht irgendwie bekannt vorkommen. Aber wenn uns der Einsatz von Software-Engineering hilft, die Projekte planbar und kontrolliert durchführbar zu machen, warum laufen dann so viele Projekte so schrecklich daneben? Als ich mein erstes Projekt selber leitete, war die Sache für mich klar: Die anderen wendeten die Techniken halt nicht konsequent genug an. Also bestand ich darauf, dass saubere Anforderungsdokumente erstellt werden, es wurde ein hundertseitiges Architekturpapier geschrieben, Testfallspezifiktionen, Werkzeuge, um aus allen diesen Dokumenten automatisch Code erzeugen, und ein ausgefeiltes Konfigurationsmanagement aufgebaut. Nach zwei Jahren bedankte sich der Auftraggeber für meine Bemühungen und entzog meinem Arbeitgeber den Auftrag. In dieser Zeit waren tolle Konzepte entstanden, aber noch keine Zeile Code für das Endsystem, weil der verwendete Prozess die Codierung erst für ein spätere Phase vorgesehen hatte. Natürlich fehlte es auch nicht an hilfreichen Kollegen, die mir mit guten Ratschlägen zur Seite standen:

- „Bei hundert Seiten Architektur hat es halt am nötigen Pragmatismus gefehlt." Aber worin hätte der Pragmatismus bestanden? Ohne Architektur starten? Wohl kaum!
- „Die Objektorientierung ist halt nicht ausgereift für große Projekte." Nicht besonders hilfreich, wenn man bedenkt, dass die vorgesehene grafische Oberfläche mit funktionaler Technik überhaupt nicht machbar gewesen wäre.

Aus heutiger Sicht lag mein Fehler wirklich an zu großem Perfektionismus, aber nicht in der Ausarbeitung des Designs, sondern beim Verfolgen eines schwergewichtigen Wasserfall-Prozesses.

Der Angriff auf die Grundfesten

Das Projekt ist gescheitert, *weil* ich mich an den Prozess gehalten habe? Diese Aussage ist provokant und bedarf daher einer genaueren Erklärung. Eine der Grundannahmen des klassischen Software-Engineering betrifft die Fehlerkosten. Je früher im Prozess ein Fehler gemacht wird, um so teurer wird er im Laufe des Projekts. Oder anders formuliert: Was du am Anfang sagst, muss richtig sein, weil du es nicht mehr ändern kannst. Und weil diese Aussage so wichtig ist, noch eine dritte Formulierung des gleichen Axioms: Einmal geschriebene Software ist nicht mehr änderbar. Die meisten Prozesse sind daher darauf ausgelegt, Fehler möglichst früh zu vermeiden und alle Fehler auszumerzen, bevor die Programmierung beginnt. Bei dem Entwicklungsprozess eines Kunden habe ich einmal 211 Reviews unterschiedlichster Dokumente gezählt, bevor die Programmierung starten konnte. Meines Wissens ist dieser Ansatz nie umgesetzt worden.

Genau betrachtet leitet sich aus dem Axiom der „Nicht-mehr-Änderbarkeit" die Parallelität zwischen der Bauindustrie und dem Software-Engineering ab: Ein chemischer Reaktionsturm, der einen Meter zu weit links gebaut wird, kann die gesamte Fabrik hinfällig machen. Ihn nachträglich zu verschieben, kann Millionen kosten. Aber gilt das wirklich auch für Software? Was wäre, wenn Software einfach änderbar wäre? Wie sähe ein Prozess aus, wenn man keine Angst mehr vor Änderungen haben müsste?

Ballast abwerfen

Zunächst einmal müsste man versuchen, alle Änderungshindernisse zu beseitigen, die aus dem Prozess selbst kommen. Sind 211 Dokumenttypen erst einmal geschrieben, verbietet sich jede Änderung von selbst, weil dann ja alle 211 Dokumenttypen konsistent nachgezogen werden müssten. Will ich leichte Änderbarkeit erreichen, muss ich also möglichst wenig Quellen haben, im besten Fall sogar nur eine einzige. Da für ein lauffähiges System zumindest Code notwendig ist, sollte man also versuchen, weitere Dokumente möglichst zu vermeiden, ohne dabei in die gute alte Zeit des chaotischen Hackens zurück zu fallen. Dies ist die Grundidee der sogenannten „leichtgewichtigen Prozesse". Die beiden bekanntesten Vertreter sind der „Crystal Clear Process" von Alistair Cockburn (vgl. [Coc02]) und das „Extreme Programming" von Ward Cunningham, Kent Beck und Ron Jeffries (vgl. [Bec00]).

Zur Erklärung möchte ich hier aus Alistair Cockburns „Manifest der Softwareentwicklung" zitieren:**

„Softwareentwicklung ist ein kooperatives Spiel, in dem die Mitspieler Markierungen und Requisiten verwenden, um sich gegenseitig und selbst zu erinnern und über den nächsten Spielzug zu informieren und

zu inspirieren. Das Ziel des Spiels ist das lauffähige Softwaresystem; die Überbleibsel des Spiels sind eine Reihe von Markierungen, welche die Spieler des nächsten Spiels informieren und ihnen helfen. Das nächste Spiel ist die Änderung oder der Ersatz des Systems, aber auch die Schaffung eines Nachbarsystems."

Dieses Manifest im Detail zu analysieren, würde den Rahmen dieses Artikels sprengen, daher sei hier lediglich auf die in der Fußnote angegebene Web-Adresse verwiesen. Ich möchte hier nur im Folgenden Aspekte herausgreifen, die meines Erachtens für das Verständnis leichtgewichtiger Prozesse von Bedeutung sind.

Softwareentwicklung ist ein kooperatives Spiel

Im Gegensatz zu den üblichen Wettkampfspielen gibt es bei kooperativen Spielen keine Gewinner oder Verlierer, sondern die ganze Gruppe arbeitet auf ein gemeinsames Ziel hin. Klettern oder Musizieren sind solche Spiele, aber auch der große Fundus an „New Games", die in allen Spielarten der Pädagogik zu finden sind. Unabhängig von dem Ziel, das es zu erreichen gilt, ist diesen Spielen eines gemeinsam: Sie erfordern ein hohes Maß an Kommunikation und Interaktion zwischen den Spielern und der Erfolg hängt von jedem einzelnen ab. Bei manchen dieser Spiele, wie zum Beispiel beim Klettern, hängt sogar das Leben der Mitspieler am reibungslosen Zusammenspiel aller Beteiligten.

Auch in der Softwareentwicklung hängt der Erfolg des Projekts von der reibungslosen Kommunikation zwischen den Beteiligten ab. Wenn die Entwickler die Probleme der Anwender nicht verstanden haben, ist das Projekt zum Scheitern verurteilt. Das gleiche gilt, wenn sie die Architektur nicht begriffen haben. Um diese Kommunikation sicherzustellen, setzen schwergewichtige Prozesse auf Dokumente, die geschrieben, einem Review unterzogen und schließlich ausgetauscht werden. Allerdings sind geschriebene (oder auch gezeichnete) Dokumente eine sehr schmalbandige Art zu kommunizieren. Um nützlich zu sein, müssen Sie sehr präzise und unmissverständlich sein. Leider ist der dafür erforderliche Aufwand in vielen Projekten unwirtschaftlich und wird deswegen nicht betrieben.

Leichtgewichtige Prozesse setzen dagegen auf direkte zwischenmenschliche Kommunikation im Gespräch oder in Teammeetings. Das bringt mindestens drei Vorteile:

- Es ermöglicht sofortige Rückfragen.
- Alle Beteiligten können zu einem sehr frühen Zeitpunkt einbezogen werden.
- Die gesamte Bandbreite der Kommunikation kann genutzt werden, von der Körpersprache über Tafelskizzen bis hin zur gemeinsamen Arbeit an Entwicklungswerkzeugen.

Erkauft werden diese Vorteile durch schlechtere Kontrollierbarkeit und eine Größenbegrenzung des Teams, auf die ich später noch kommen werde.

Ziel ist das lauffähige Softwaresystem

Was wie eine Selbstverständlichkeit klingt, kann nicht oft genug hervorgehoben werden. Vielleicht sollte man zur Verdeutlichung schreiben: Ziel ist einzig das lauffähige Softwaresystem. Ich habe dutzende von Projekten gesehen, in denen das Ziel augenscheinlich darin bestand, eine tolle Entwicklungsumgebung zu schaffen, schöne Dokumente zu schreiben oder eine besonders leistungsfähige Middleware zu bauen. Für Firmen, die Entwicklungsumgebungen oder Middleware als Produkte bauen, oder für Autoren sind das die richtigen Ziele. Bei den meisten Projekten verdient aber der Auftraggeber nicht sein Geld mit dem Vertrieb von Entwicklungsumgebungen, Middleware oder Büchern. Die Software ist ein Werkzeug für ihn, das vor allem Geld kostet. Sie muss so schnell wie möglich in Produktion gehen, weil dadurch ein Wettbewerbsvorteil entstehen könnte (oder ein Wettbewerbsnachteil ausgemerzt wird), und sie sollte auch möglichst wenig Folgekosten in Betrieb und Wartung produzieren. Warum Millionenbeträge in die Eigenentwicklung technischer Infrastruktur oder (meist unlesbarer) Dokumente investieren, wenn ich mittlerweile sehr gute Infrastrukturen kaufen kann und die Dokumente durch intensive Kommunikation im Team weitgehend reduzieren kann? Wenn es die benötigte Infrastruktur auf dem Markt noch nicht gibt, kann man häufig durch geringfügiges Abspecken der Anforderungen auf käufliche Umgebungen ausweichen.

In dem von mir oben erwähnten Projekt war zum Beispiel eine der Anforderungen, die grafische Oberfläche plattformübergreifend zu bauen. Der Kunde wollte sich die Option offen halten, später noch von Windows auf Unix umzustellen. Als alter Unix-Freund nahm ich die Anforderung gerne an und produzierte damit einen Aufwand von mehreren Bearbeiterjahren, da wir viele Entwicklungswerkzeuge nicht mehr nutzen konnten und zum Teil selbst schreiben mussten (Java war damals noch nicht am Markt). Es wäre vermutlich billiger gewesen, im Falle einer Migration das System einfach noch mal neu zu schreiben. „You Aren't Gonna Need It" (auf dt. „Du brauchst es eh' nicht") ist eines der schönsten Bonmots von Kent Beck. Neunzig Prozent der Dinge, die als „vielleicht müssen wir später mal..." in Softwareprojekte hineinkriechen, werden niemals benötigt. Wenn man sich diese Aufwände spart und dafür bei jeder Anforderung, die doch noch kommt, den dreifachen Aufwand hat, kommt unter dem Strich noch immer eine Ersparnis von siebzig Prozent zustande! Aus meiner persönlichen Erfahrung erschei-

** Die englische Originalfassung ist verfügbar unter http://members.aol.com/acockburn/manifesto.html. Dort findet sich auch eine ausführliche Diskussion des Manifests. Die Übersetzung und Verwendung hier erfolgte mit freundlicher Genehmigung von Alistair Cockburn.

GEBIT
Wir realisieren E-Commerce/E-Business Anwendungen auf Basis von Java2EE, EJB, JSP, Servlets...
OOAD
EJB
JSP
J2EE
JMS
...
Java-Profis
Wenn es um die Realisierung umfangreicher IT-Projekte auf Basis moderner OO-Technologien geht, vertrauen viele erfolgreiche Unternehmen seit 1991 auf GEBIT - und das nicht nur in Deutschland. Bereits seit 1996 setzen wir auf Java!
Auf Sie warten innovative Projekte, die Mitarbeit in einem professionellen Team und jede Menge
fun@work
www.gebit.de
Gleich online bewerben!
Tel.: 030 / 896 663 00
The (e)-Technology People

nen mit bei einer einigermaßen tragfähigen Architektur diese Zahlen durchaus realistisch. Ziel ist also das lauffähige Softwaresystem – und sonst nichts. Gemacht wird nur, was man dafür benötigt. Alles andere kann auf später warten, selbst wenn es dann etwas teurer wird.

Der Zweck heiligt die Mittel

Außer Code dient alles, was innerhalb eines Projekts entsteht, nur dem Zweck, den Teammitgliedern zu helfen und sie zu unterstützen. Mit anderen Worten: Der Zweck heiligt die Mittel. Wenn das Team der Ansicht ist, dass es besser arbeitet, wenn es ein Klassenmodell mit kleinen, gelben Klebezetteln und Wollfäden an der Wand entstehen lässt, ist das genauso gut, als wenn dafür ein CASE-Werkzeug eingesetzt wird oder das Klassenmodell nur im Code-Browser existiert. Natürlich vertreten Werkzeughersteller hier eine etwas andere Position, doch darf man sie hier getrost als voreingenommen betrachten. Wichtig ist lediglich, dass das Team seine Arbeit machen kann und regelmäßig die Möglichkeit hat, darüber zu reflektieren, wie es die Arbeit macht. Welche Dinge schriftlich kommuniziert werden und auf welche Weise, hängt dabei stark vom jeweiligen Team ab. Es ist kaum vorstellbar, dass eine irgendwie geartete Methodengruppe oder auch ein Projektleiter oder ein Berater hier mehr leisten können, als dem Team Optionen und Alternativen aufzuzeigen. Die Spielregeln kooperativer Spiele müssen von allen Mitspielern entworfen und getragen werden. Das Management muss also die Entscheidung des Teams nach außen durchfechten. Vom Controlling verordnete Klebezettel als billiger Ersatz für CASE-Werkzeuge wirken letal.

„Moment mal" werden erfahrene Projektleiter und Anwendungsentwickler jetzt einwenden, „Was ist mit der Wartungsdokumentation? Ich kann doch schlecht die Tapete von der Wand reißen und archivieren, nur weil ein paar Klebezettel darauf mein System dokumentieren." Das ist ohne Einschränkung korrekt. Aber Hand aufs Herz: Wie viele Projekte haben Sie in der Wartung, in denen die Dokumentation wirklich so aktuell ist, dass sie von der Wartungsmannschaft genutzt und aktualisiert wird? Ich würde 1:10 wetten, dass weniger als 10% der Anwendungen in diesem Zustand sind, und könnte mit den Wetterlösen vermutlich einen äußerst attraktiven Fond begründen. In den allermeisten Fällen werden Dokumente früher Phasen zwar geschrieben, aber spätere Änderungen werden nicht mehr eingepflegt, wenn der Terminplan drückt.

„Im Code steckt die Wahrheit" ist eine alte Weisheit, an deren Gültigkeit alle Fortschritte des Software-Engineering wenig geändert haben. Anstatt gegen diese Wahrheit anzukämpfen, kann man sie auch als gegeben hinnehmen. In diesem Fall würde man vielleicht ein Fünftel des Aufwands, der bei schwergewichtigen Prozessen in die Dokumentation gesteckt wird, in die Vereinfachung des Codes stecken. Was ich an Dokumentation wirklich brauche, ist ein kurzer Überblick über die Funktionsweise und die wesentlichen Elemente des Systems; alles andere sollte über Code-Browser zur Verfügung stehen. Ein solcher Überblick – nennen Sie ihn Architektur, Vision oder wie immer Sie wollen – sollte auf zehn Seiten passen oder in einer halben Stunde erklärt sein.

Ehrliche Werbung

„Prima, endlich können wir uns diese lästigen Analyse- und Dokumentationsarbeiten sparen und direkt anfangen, das zu tun, was uns am meisten Spaß macht: Hacken!" Diese Schlussfolgerung ist nicht selten. Ich erinnere mich an eine Diskussion, die ich im Oktober mit Kent Beck hatte, ob der aktuelle Hype um „eXtreme Programming" (XP) den leichtgewichtigen Prozessen mehr Nutzen oder Schaden bringen würde (siehe auch Beitrag auf S. 28). Beck räumte ein, dass auch er immer wieder „Erfahrungsberichte" hören würde, in denen stolz verkündet wird: „Wir machen XP, aber das Pair Programming lassen wir weg und das Planungsspiel funktioniert bei uns nicht so richtig und das mit den automatischen Tests ist bei uns früher schon mal gescheitert und der Anwender im Team ist uns zwar versprochen worden, kommt aber erst im nächsten Jahr!" Für den geschulten Blick bleibt hier nur eine Diagnose: Da hat jemand die Idee leichtgewichtiger Prozesse nicht richtig verstanden. Sie sind keineswegs eine Aufforderung, wieder in die dunklen Zeiten des Hackens zurückzufallen und alles über Bord zu werfen, was in den letzten zwanzig Jahren entwickelt wurde. Leichtgewichtige Prozesse verlangen vielmehr ein ausgesprochen diszipliniertes Vorgehen, sie besitzen nur sehr wenig Redundanz und sind daher anfälliger gegen Prozessabweichungen als schwergewichtige Prozesse. Dass man bei konsequenter Anwendung dieser Prozesse auch auf manche lästige Dokumentationsarbeit verzichten kann, ist eher die Ernte, die man einfährt für reibungslos funktionierendes Konfigurationsmanagement, voll automatisierte Tests und andere Maßnahmen.

Was ich in dieser Diskussion also bisher stillschweigend unterschlagen habe, sind drei Voraussetzungen, die absolut notwendig sind, damit ein solches Vorgehen trägt:

- Die Rückkopplungsschleifen müssen kurz genug sein.
- Im Team muss Kommunikation möglich sein.
- Das Team muss seinen Job beherrschen.

Zumindest die beiden ersten Voraussetzungen sind in vielen klassischen Projekten nicht gegeben.

Möglichkeit, zu lernen

Kurze Rückkopplungsschleifen benötigen alle Beteiligten, um den Erfolg des Projekts sicherzustellen. Das Team benötigt Erfahrung, um einschätzen zu können, welche Verfahren funktionieren und welche nicht. Das Management benötigt früh Rückkopplung, um sicher zu sein, dass das Projekt nicht aus dem Ruder läuft. Die Anwender brauchen schließlich frühzeitig lauffähige Systeme, um zu sehen, ob das System wirklich das liefert, was sie benötigen. Für alle drei Zwecke ist laufender Code das Mittel der Wahl. Ein komplett durchlaufener Entwicklungszyklus gibt dem Team überhaupt erst die Möglichkeit, die Brauchbarkeit einzelner Verfahren zu beurteilen. Für das Management ist ein laufendes System ein ziemlich unbestechlicher Indikator dafür, dass etwas läuft. Angeblich sind Softwaresysteme zu 80% der vorgesehenen Zeit zu 80% fertig. Die restlichen 20% benötigen wiederum 80% der Zeit. Ein laufendes System ist per Definition fertig, kleinere Nachbesserungen dienen dazu, weitere Stufen darauf setzen zu können, und zählen also schon zur nächsten Stufe. Leichtgewichtige Prozesse verlangen also zwingend ein iteratives Vorgehen.

Es fehlen noch die Anwender. Wenn es einem Experten schon schwer fällt, die Brauchbarkeit eines Systems allein anhand von Dokumenten zu beurteilen, um wie viel schwerer tut sich dann ein Laie, der lediglich die Arbeit gemacht haben möchte? Ein laufendes System gibt den Anwendern die besten Möglichkeiten, Änderungswünsche zu formulieren und sie mit den Entwicklern durchzusprechen. Schließlich ist Software änderbar.

Miteinander reden

Ein anderes Problem ist die Kommunikationsfähigkeit eines Teams. Damit ein Team miteinander reden kann, darf es nicht zu groß sein und muss beieinander sitzen – beides Voraussetzungen, die häufig missachtet werden. Es gibt eine obere Grenze für die Teamgröße, ab der die direkte Kommunikation nicht mehr funktioniert. Mathematisch gesehen steigt die Anzahl der notwendigen Kommunikationsverbindungen mit dem Quadrat der Anzahl der Teammitglieder. Die Zahl, wo der Rubikon überschritten wird, schwankt zwischen 8 und 15 Personen – je nach Quelle. Ab dieser Grenze ist es nicht mehr möglich, alle Teammitglieder auf dem Laufenden zu halten und ohne Rundschreiben, schriftliche Vorgaben und Ähnlichem zu arbeiten. In Abwandelung eines Wortes von Tom DeMarco könnte man sagen, ein Team von 40 Personen ist nur unwesentlich ineffizienter als eines von zehn. Andere Redensarten stützen diese Überlegung. „Wenn man ein verspätetes Projekt personell aufstockt, wird es noch später", ist eine davon.

Was aber ist mit den vielen Projekten, an denen 100 und mehr Personen arbeiten? Nun, die erschreckend niedrige Erfolgsquote solcher Projekte bestätigt diese Überlegungen eher, als ihnen zu widersprechen. Leider ist damit aber nicht gesagt, dass eine radikale Verkleinerung des Teams immer die richtige Lösung wäre. Manche Systeme, wie zum Beispiel Telekommunikationsanlagen, sind einfach nicht mit kleineren Teams zu machen. Man kann natürlich kleine Teilteams bilden und versuchen, die Schnittstellen zwischen ihnen zu minimieren. Aber diese Schnittstellen müssen sauber dokumentiert und schriftlich fixiert sein, sonst läuft das System aus dem Ruder. Dennoch gibt es viele Fälle, wo ein Team von acht oder zehn versierten Softwareentwicklerinnen und -entwicklern bei weitem rentablere Ergebnisse erbracht hätten als große Teams. Als Paradebeispiel möchte ich nur das Gehaltssystem von Chrysler nennen, das durch eine Schlankheitskur von 26 auf acht Entwickler innerhalb von zwei Jahren in Produktion gesetzt werden konnte.

Neben der reinen Größe ist aber auch die räumliche Nähe des Teams entscheidend. Bei einem meiner Kunden wurden einmal wegen eines Umzugs vier Kolleginnen und Kollegen vom restlichen Team räumlich getrennt. Die Folge waren erhebliche Turbulenzen im Projektablauf. Es dauerte fast drei Monate, bis sich das Team auf die neue Situation eingestellt und verstärkt auf E-Mail und eine Fehlerdatenbank gesetzt hatte. Die Vorstellung, man könne Softwareentwicklung rund um den Erdball verteilen, um die einzelnen Zeitzonen oder einfach die niedrigen Löhne in manchen Ländern besser nutzen zu können, verkennt diesen Kommunikationsaspekt völlig. Wer einmal geklettert ist, weiß, wie schwierig das Sichern wird, wenn die kletternde Person auf einmal außer Sichtweite gerät. Oder können Sie sich ein Konzert vorstellen, bei dem die Streicher in Japan sitzen, die Bläser aber in London und die Paukisten in Lima? Versuche in dieser Richtung hatten eher medialen als musikalischen Wert.

Ein dritter Aspekt ist die persönliche Kommunikationsfähigkeit der einzelnen Teammitglieder. Kommunikation ist Handwerk. Zwar lernen wir alle einen guten Teil des Handwerks schon in den ersten Lebensjahren, so wie die meisten Kinder auch Malen oder Schnitzen lernen. Professionelle Maler oder Schnitzer haben aber meist auch eine weitere Ausbildung genossen. Wenn Softwareentwicklung Kommunikation ist, müssten professionelle Softwareentwickler dann nicht auch professionelle Kommunikatoren sein? Wenn ich mir überlege, wie viel Zeit und damit Geld und Aufwand in vielen Projekten einfach durch Missverständnisse und schlechte Kommunikation verloren gehen, ist es ein Wunder, dass viele Unternehmen noch immer das Erlernen von Programmiersprachen als die Schlüsselqualifikation eines Softwareentwicklers sehen und nicht das Erlernen professioneller Kommunikation. Da wird eher auf schlecht geschriebene Dokumente zurückgegriffen, die zu astronomischen Kosten für den Papierkorb geschrieben werden.

Garbage in, Garbage out

Als letzte Voraussetzung fehlt noch die schlichte Qualifikation des Teams. „Und hier habe ich noch zwei Wochen geplant, um das Team in Objektorientierung zu schulen." Welcher Berater hätte solche Sätze noch nicht gehört? Was würden Sie mit einem Bauleiter machen, der Ihnen erklärt, die Maurer wären gerade noch auf einem zweiwöchigen Kurs über Hausbau? Ich persönlich würde mir schleunigst eine neue Baufirma suchen! Es ist richtig, dass gute Softwareentwickler mit der entsprechenden Grundqualifikation sehr schnell neue Technologien erlernen können. Es ist aber auch richtig, dass schlechtes Design eine der häufigsten Ursachen für gescheiterte Projekte ist. Die Vorstellung, man könne mit Hilfe eines wie auch immer gearteten Prozesses aus einem schlechten Team gute Arbeit holen, ist – vorsichtig gesagt – naiv.

Über sieben Brücken musst Du geh'n

So lautet ein alter Rocktitel der Gruppe Karat. In der Softwareentwicklung haben wir schon einige Brücken hinter uns: Wir haben die Mathematik als unentbehrliche Hilfe beim Finden von Algorithmen erkannt, wir haben Programmiersprachen entwickelt, wir haben gelernt, die Komplexität zu beherrschen, wir haben von anderen Ingenieursdisziplinen das Management großer Projekte gelernt und wir haben unentbehrliche Werkzeuge entwickelt, die uns dabei helfen. Dies reicht von Compilern über das Konfigurationsmanagement und automatische Tests bis hin zu kompletten integrierten Entwicklungsumgebungen und zahlreichen Frameworks. Es wird Zeit, dass wir von dieser langen Reise Ballast abwerfen und die nächste Brücke suchen. Vielleicht besteht sie aus der Erkenntnis, dass Software eben keine Fabrik ist, sondern änderbar und dass sie vor allem in einem kommunikativen Prozess erstellt wird.

Literatur

[Bec00] K. Beck, Extreme Programming Explained – Embrace Changes, Addison-Wesley, 2000

[Brö95] A.-P. Bröhl, W. Dröschel, Das V-Modell – Der Standard für die Softwareentwicklung mit Praxisleitfaden, R. Oldenbourg Verlag, 1995

[Coc02] A. Cockburn, Crystal Clear: A Human-Powered Methodology for Small Teams, Addison-Wesley (erscheint voraussichtlich 2002, Online-Version siehe http://members.aol.com/humansandt/crystal/clear)

[Jac99] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999

[You97] E. Yourdon, Death March – The Complete Software Developer's Guide to Surviving Mission Impossible Projects, Prentice Hall, 1997

Bruhn, M.: Marketing. 5. Aufl., Wiesbaden 2001.
Hill, W.: Marketing. Band 1. 5. Aufl., Bern/Stuttgart 1984.
Homburg, C./Krohmer, H.: Marketingmanagement. Wiesbaden 2003.
Kleinhückelskoten, H.-D./Holm, J.-M.: Marketing-Mix. Köln 2000.
Kotler, P./Bliemel, F.: Marketing-Management. 10. Aufl., Stuttgart 2001.
Kotler, P. et al.: Grundlagen des Marketing. 2. Aufl., München et al. 1999.
Meffert, H.: Marketing. 9. Aufl., Wiesbaden 2000.
Nieschlag, R./Dichtl, E./Hörschgen, H.: Marketing. 19. Aufl., Berlin 2002.
Pepels, W.: Marketing. 4. Aufl., München/Wien 2004.
Weis, H.C.: Marketing. 13. Aufl., Ludwigshafen 2004.

BASISWISSEN WIRTSCHAFTSINFORMATIK

Objektorientierung bei der Softwareentwicklung

Die Objektorientierung spielt eine wichtige Rolle bei der Software-Entwicklung. Mittlerweile beruhen viele Anwendungssysteme auf dieser Vorgehensweise, die **andere Programmierkonzepte** weitgehend **abgelöst** hat. Software-Aufgaben werden dadurch beschrieben, dass die beteiligten Objekte und ihre Kommunikationsbeziehungen untereinander abgebildet werden. Ein wesentliches Merkmal ist die Zusammenfassung von Daten und Funktionen zu Objekten.

Einordnung in die Sprachgenerationen

Heute werden sechs Generationen von Programmiersprachen unterschieden:

- Zur ersten Generation gehören plattformspezifische **Maschinensprachen**. Die als Abfolge von Dualzahlen formulierten Befehle werden direkt vom Rechner ausgeführt, womit eine hohe Verarbeitungsgeschwindigkeit erreicht wird. Die Programme können jedoch nicht von anderen Rechnertypen verwandt werden.
- **Assemblersprachen** sind eine Weiterentwicklung. Mit ihnen werden die dualen Operationscodes der Befehle und die Adressen der Variablen aus der Maschinensprache mit Hilfe gedächtnisunterstützender Kürzel verständlich dargestellt. Ab dieser Generation müssen die Anweisungen vor der Ausführung in die Maschinensprache übersetzt werden. Die Vorteile von Assemblern gegenüber der Maschinensprache liegen in der wesentlich vereinfachten Programmentwicklung. Trotzdem wird bei gleicher Abarbeitungsgeschwindigkeit nur unwesentlich mehr Speicherplatz benötigt. Beide Generationen sind nicht sehr benutzerfreundlich und daher fehleranfällig.
- Programmiersprachen der dritten Generation werden als **prozedurale oder problemorientierte Sprachen** bezeichnet. Sie sind stärker an den Bedürfnissen der Aufgabenstellungen ausgerichtet. Beispielsweise erleichtern Sprachkonstrukte für Kontrollstrukturen den Aufbau komplexer Systeme. Daneben werden elementare und komplexe Datentypen unterstützt. Vertreter dieser Sprachgeneration sind unter anderem Basic, C, COBOL, Fortran und Pascal. Problemorientierte Sprachen erlauben die Beschreibung der Abarbeitungsschritte einer Aufgabe in Form von Algorithmen. Durch Sprünge, Schleifen oder Verzweigungen besteht bei großen Programmen die Gefahr der Unübersichtlichkeit. Die Plattformunabhängigkeit erlaubt den Einsatz auf verschiedenen Rechnertypen. Der Quellcode muss dabei auf den verschiedenen Plattformen jeweils neu übersetzt werden. Als Nachteil erweist sich die schlechte Hardware-Ausnutzung und die damit verbundenen längeren Programmlaufzeiten.
- **Deklarative Sprachen** bilden die vierte Sprachgeneration. Der wesentliche Unterschied zu prozeduralen Sprachen: Es wird beschrieben, welches Ziel angestrebt wird, hingegen nicht, wie die Aufgabe in einzelnen Schritten zu lösen ist. Der Vorteil solcher Sprachen ist, dass sie relativ einfach zu erlernen sind, allerdings sind ihre Anwendungsbereiche beschränkt. Zu diesen Sprachen gehören beispielsweise SQL zur Datenbankabfrage und LabVIEW zur grafischen Programmierung.
- Die fünfte Generation sind **Sprachen der künstlichen Intelligenz.** Sie erfüllen die Anforderungen wissensbasierter Systeme. An Stelle von Verarbeitungsvorschriften werden Regeln und Fakten hinterlegt, wodurch die Programmausführung bestimmt wird. Zu diesen Sprachen rechnen beispielsweise Prolog und LISP.
- **Objektorientierte Sprachen** werden der sechsten Generation zugeordnet. Die Programme bestehen aus miteinander kommunizierenden Objekten, was ihre Funktionalität ausmacht. Im Gegensatz zu den prozeduralen Sprachen findet keine Zerlegung in Teilfunktionen, sondern in die beteiligten Objekte statt. Damit erlauben objektorientierte Programme eine natürliche Beschreibung realer Abläufe. Smalltalk und Java sind Beispiele dafür.

Grundbegriffe

Objekte repräsentieren Dinge aus der realen Welt. Sie verfügen über Eigenschaften **(Attribute)** und Operationen **(Methoden)**, um die Zustände dieser Eigenschaften zu ändern. Objekte werden durch ihren Zustand (Ausprägung aller Attribute), ihr Verhalten (Reaktionen des Objekts auf bestimmte Methodenaufrufe) und ihre Identität (Eindeutigkeit des Objekts) charakterisiert. Objekte treten untereinander in Beziehung, indem sie Botschaften austauschen. Diese **Nachrichten** werden durch Methodenaufrufe realisiert. Ein Objekt kann also nicht direkt die Daten eines anderen Objektes verändern, es muss dessen entsprechende Methode aufrufen können. Diese Nachrichten müssen alle Eingangsparameter (Eingabewerte) und Ausgangsparameter (Rückgabewerte) umfassen, die für die Abarbeitung benötigt werden.

Objekte mit ähnlichen Eigenschaften und Operationen können zu einer **Klasse** zusammengefasst und entsprechend abstrahiert werden. Eine Klasse repräsentiert damit den Bauplan ihrer Objekte. Eigenschaften und Methoden sind abstrakt in Klassen festgelegt. Eine **Instanz** ist ein nach den Vorschriften der Klasse konkret erzeugtes Objekt. So definiert etwa die Klasse „Fahrrad“ allgemein die Eigenschaften („Satteltyp“, „Farbe“, ...) und Methoden („beschleunigen()“, „bremsen()“, „lenken()“, ...) eines Fahrrads. Ausgehend von diesem Bauplan

STICHWORT DES MONATS

Defizitverfahren

Nach dem Stabilitäts- und Wachstumspakt gilt ein Staatsdefizit als übermäßig, wenn es 3% Prozent des Bruttoinlandprodukts überschreitet. Stellt der ECOFIN-Rat nach Artikel 104 c (6) EGV auf Empfehlung der Kommission solch ein Defizit fest, tritt ein strikter Zeitplan für die vorgeschlagen Korrekturmaßnahmen in Kraft, an dessen Ende Sanktionen in Form von zunächst zinslosen Einlagen und schließlich Strafen in Milliardenhöhe stehen können.

Im Jahr 2002 wies Deutschland ein Staatsdefizit in Höhe von 3,5% des Bruttoinlandsprodukts (BIP) auf. Der ECOFIN-Rat stellte daraufhin auf Empfehlung der Kommission im Januar 2003 ein „übermäßiges Defizit" fest und forderte das Land auf, innerhalb von vier Monaten Konsolidierungsmaßnahmen im Umfang von 1% des BIP vorzunehmen, damit im kommenden Jahr die Defizitgrenze von 3% des BIP eingehalten werden könnte. Im Mai 2003 bestätigte die Europäische Kommission, dass Deutschland die erforderlichen Maßnahmen beschlossen hatte. Doch in der Herbstprognose 2003 zeichnet sich ab, dass Deutschland die Defizitgrenze im Jahre 2004 abermals überschreiten würde. Die Kommission empfahl daraufhin dem ECOFIN-Rat, Deutschland wegen unzureichender Konsolidierungsmaßnahmen (als letzten Schritt vor der Verhängung von Sanktionen) in Verzug zu setzen. In der Sitzung des Rats am 25. November 2003 fanden diese Empfehlungen jedoch nicht die erforderliche Mehrheit. Statt dessen empfahl der Rat, das deutsche konjunkturbereinigte Defizit im Jahr 2004 um 0,6% und in den folgenden Jahren um jeweils 0,5% zu reduzieren. Die Kommission hielt diesen Aufschub nicht mit Artikel 104 des EG-Vertrags vereinbar und erhob im Januar 2004 Klage beim Europäischen Gerichtshof.

Der milde Beschluss des ECOFIN-Rats wurde gewöhnlich damit erklärt, dass einige Mitglieder rein taktisch votiert hätten, da sie selbst aktuelle oder potenzielle Sünder seien (insbesondere Frankreich). Allerdings können auch makroökonomische Interdependenzen zur Begründung herangezogen werden: Die Budgetbestimmungen des Stabilitäts- und Wachstumspakts beruhen nämlich vor allem auf der Vorstellung, das Defizit eines Mitgliedslandes schädige die Partnerländer, weil dadurch auf dem gemeinsamen Kapitalmarkt eine Zinssteigerung hervorgerufen werde. Bei schwacher wirtschaftlicher Aktivität und reichlicher Liquiditätsversorgung, wie sie gegenwärtig in Europa herrscht, übt das Deficit Spending eines Landes jedoch einen positiven Effekt auf die Partnerländer aus, da über die Wirkung des Außenhandelsmultiplikators deren Einkommen zunimmt und ein damit steigendes Steueraufkommen ihr Haushaltsdefizit verringert. Im Umkehrschluss würde ein diskretionärer Defizitabbau Deutschlands in den Partnerländern eine eher schwächere Beschäftigungslage und eine endogene Defiziterhöhung bewirken. Ist also der Konflikt des Rats mit der Kommission ein erstes Aufbegehren gegen die „Erstarrung der Europa-Politiker in einer präkeynesianischen Orthodoxie" (Paul de Grauwe)?

Prof. Dr. Anton Konrad, München

kann eine Instanz erzeugt werden, die spezifische Ausprägungen der in der Klassendefinition angegebenen Attribute aufweist (Satteltyp Kunststoff, Farbe rot, ...).

Konzepte

— **Vererbung**: Neue Klassen können auf der Grundlage von bereits vorhandenen Klassen definiert werden. So können einerseits verschiedene Klassen zu neuen übergeordneten Klassen zusammengefasst werden **(Generalisierung)**. Andererseits besteht die Möglichkeit, aus einer bestehenden Klasse detailliertere Unterklassen abzuleiten **(Spezialisierung),** womit eine **hierarchische Struktur** entsteht. Vererbung bedeutet in diesem Zusammenhang, dass alle Attribute und Methoden einer übergeordneten Klasse automatisch in den untergeordneten Klassen implementiert sind und nicht neu definiert werden müssen. Vererbende Klassen werden als Basis- oder Superklassen, erbende Klassen als abgeleitete Klassen oder Subklassen bezeichnet. Da abgeleitete Klassen alle Attribute und Methoden ihrer Basisklasse erben, wirken sich Veränderungen in den Basisklassen direkt auf alle Subklassen aus. Auf diese Weise wird die Wartung komplexer Systeme erleichtert. Geerbte Eigenschaften und Operationen können übernommen, verändert oder erweitert werden. Abhängig davon, ob eine Subklasse von einer oder mehreren Basisklassen abgeleitet wurde, unterscheidet man Einfach- und Mehrfachvererbung. Da Mehrfachvererbung zu Konflikten bei der Namensgleichheit von Attributen oder Methoden führen kann, ist dieses Konzept nicht in allen objektorientierten Sprachen vorgesehen.

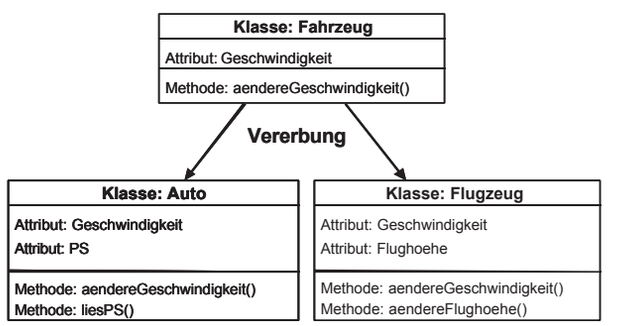


Abb. 1: Vererbung

Wie das Beispiel in Abb. 1 zeigt, wurden in der Klasse „Fahrzeug" das Attribut „Geschwindigkeit" und die Methode „aendereGeschwindigkeit()" implementiert. Durch Vererbung stehen diese automatisch auch in den Subklassen „Auto" und „Flugzeug" zur Verfügung und müssen nicht neu definiert werden. Zusätzlich können die abgeleiteten Klassen weitere eigene Eigenschaften und Operationen implementieren.

— **Kapselung**: Der Zustand eines Objekts, d.h. die Werte seiner Attribute, kann durch andere Objekte nur verändert werden, wenn geeignete Methoden implementiert wurden. In der Objektorientierung ist somit der Zugriff auf die Daten eines Objekts nur möglich, wenn dieses Objekt die Übermittlung oder **Manipulation** von Daten durch eine entsprechende Operation zulässt. Ein direkter Zugriff von außen wird verhindert. Der indirekte Zugriff kann nur durch Nachrichtenaustausch zwischen den beteiligten Objekten und durch Aufrufe entsprechender Methoden realisiert werden. Kapselung bezeichnet somit die Zusammenfassung von Name, Zustand und Verhalten eines Objekts. Dadurch wird eine ungewollte Datenmanipulation verhindert, was die Programme sicherer und stabiler macht. Abb. 2 verdeutlicht diesen Zusammenhang. Zur Erstellung einer Liste von Klausurteilnehmern benötigt das Prüfungsamt die Matrikelnummern der betroffenen Studenten. Auf-

grund der Kapselung besteht keine Möglichkeit, diese Daten direkt auszulesen. Stattdessen wird die Methode „leseMatrikelnummer()“ der Klasse „Student“ aufgerufen, welche die Operation zum Lesen und Übermitteln des gewünschten Attributs zur Verfügung stellt.

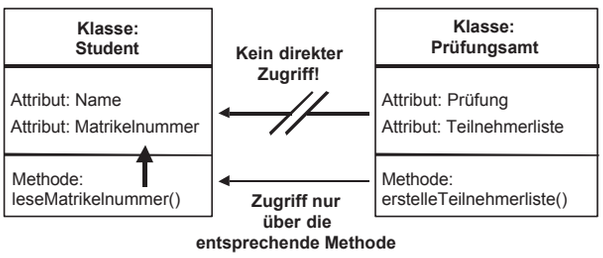


Abb. 2: Kapselung

— **Polymorphismus**: Polymorphismus bedeutet Vielgestaltigkeit. Damit wird die Tatsache bezeichnet, dass ein Objekt auf die gleiche Nachricht in unterschiedlicher Weise reagieren kann. Subklassen können die Methoden ihrer Basisklassen **überschreiben**, indem sie eine eigene (gleichnamige) Methode definieren. Die Art der Ausführung hängt somit davon ab, von welcher Klasse ein Objekt instanziiert wurde. In Abb. 1 erfolgt die Ausführung der Methode „aendereGeschwindigkeit()“ bei der Klasse „Auto“ auf andere Weise als bei der Klasse „Flugzeug“.

— **Überladen von Methoden**: Beim Überladen von Methoden existieren innerhalb einer Klasse mehrere Methoden mit dem gleichen Methodennamen. Jede dieser Methoden beschreibt ein anderes Verhalten und besitzt andere Parameter, die sich in Anzahl, Datentyp und/oder Reihenfolge unterscheiden. Die bei dem Methodenaufruf übergebenen Parameter identifizieren die gewünschte Methode. Dies soll an einem Beispiel verdeutlicht werden: Ein Objekt der Klasse „GeometrischeForm“ besitzt zwei Methoden „flaecheBerechnen()“, die der Flächenberechnung eines Kreises bzw. eines Rechtecks dienen. Die relevante Operation wird aufgrund der Ausprägung der **Parameterliste** bestimmt. Für die Berechnung der Kreisfläche muss lediglich der Radius übergeben werden (flaecheBerechnen(int Radius)), während für die Berechnung des Flächeninhalts eines Rechtecks Breite und Länge im Methodenaufruf angegeben werden müssen (flaecheBerechnen(int Breite, int Hoehe)).

— **Dynamisches Binden**: Erst wenn die Anwendung läuft, wird bestimmt, welche Methode durch das Senden einer Nachricht ausgeführt wird. Dazu wird die entsprechende Operation gesucht, die den aufrufenden Namen sowie die zugehörige Parameterliste enthält. Die Methodensuche beginnt bei der Klasse des Objekts, welches Empfänger der Nachricht ist. Ist die Methode dort nicht vorhanden, wird in den Basisklassen entlang der Vererbungshierarchie gesucht, bis die richtige Methode gefunden wurde oder ein entsprechender Fehlerhinweis aufgetreten ist.

— **Abstraktion**: In der Objektorientierung wird nicht versucht, ein vollständiges Modell des untersuchten Ausschnitts der Realität aufzubauen. Nur die für die Bewältigung der Aufgabe benötigten Teilaspekte werden abgebildet. Abstraktion bezeichnet die Beschränkung auf die für die jeweilige Sichtweise relevanten Modellelemente. Entlang der Klassenhierarchie können Eigenschaften und Methoden schrittweise konkretisiert werden. Beispielsweise definiert die abstrakte Klasse „Fahrzeug“ die Eigenschaft „Geschwindigkeit“ sowie die Methoden „beschleunigen()“ und „abbremsen()“. Wie dieses Verhalten realisiert wird, wird erst in den Subklassen festgelegt.

Objektorientierte Entwicklung

In der objektorientierten Systementwicklung werden grundsätzlich die Phasen Analyse, Design und Programmierung unterschieden. In der **objektorientierten Analyse** (OOA) werden die Anforderungen an das System ermittelt und die realen Problemstellungen in Modellen abgebildet. Bestandteile der Leistungsbeschreibung sind Klassen bzw. Objekte sowie die Beziehungen untereinander. Das Ergebnis der Analyse ist das Fachkonzept. Dieses ist der Ausgangspunkt für das **objektorientierte Design** (OOD). In dieser Phase wird das abstrakte Fachkonzept an die technischen Rahmenbedingungen angepasst. Zusätzlich werden die Benutzeroberfläche und die Datenhaltung modelliert. Anschließend erfolgt die Umsetzung durch die **objektorientierte Programmierung** (OOP) in einer entsprechenden Programmiersprache.

Die objektorientierte Systementwicklung wird durch die **Unified Modeling Language** (UML) zusätzlich unterstützt. Durch zwölf verschiedene Diagrammarten können beispielsweise Klassen, Objekte, Anwendungsfälle, Verhalten sowie Implementierung der Software abgebildet werden. Neben der Spezifikation für die Entwickler dient die Modellierung auch zur Verbesserung der Kommunikation mit dem Auftraggeber. Beteiligte Akteure und Geschäftsvorfälle sowie ihre Beziehungen können grafisch dargestellt werden. Daneben haben sich eine Reihe von **Vorgehensmodellen** entwickelt, die explizit die objektorientierte Systementwicklung unterstützen. Der gesamte Prozess von der Auftragsvergabe und Analyse bis zur Auslieferung und Wartung der Software wird durch Leitfäden und Vorgaben unterstützt.

Fazit

Die objektorientierte Systementwicklung unterscheidet sich grundlegend von den bisherigen Ansätzen. Klassenbildung, Vererbung und Abstraktion erleichtern die Entwicklung, da komplexe Aufgaben in Teilprobleme zerlegt werden können. Durch den modularen Aufbau werden die **Wiederverwendbarkeit** sowie die bessere Wartung und Pflege von Software gefördert. Die Applikationen setzen sich aus genau spezifizierten, abgeschlossenen Code-Fragmenten zusammen, die einzeln entwickelt, getestet und gegebenenfalls angepasst oder ausgewechselt werden können. Ändern sich einzelne Abarbeitungsvorschriften, hat dies nicht zwangsläufig Auswirkungen auf das gesamte System. Bleiben Eingangs- und Ausgangsparameter gleich, muss nur die Verarbeitungslogik innerhalb der betroffenen Methode angepasst werden. Die Verwendung bewährter Software-Bausteine erhöht die **Qualität** durch geringere Fehleranfälligkeit und senkt zugleich Zeitaufwand und Kosten. Zudem werden **Zuverlässigkeit** und **Robustheit** der Anwendungssysteme erhöht. Damit werden nur objektorientierte Lösungen den Anforderungen aktueller Programmieraufgaben und von Standard-Software gerecht.

Dipl.-Kfm. Martin Böhn/
Dipl.-Kfm. Alexander Hagn, Würzburg

Literaturempfehlungen:

Balzert, H.: Lehrbuch der Software-Technik. Software-Entwicklung. 2. Aufl., Heidelberg 2000.
Balzert, H.: Objektorientierung in 7 Tagen. Vom UML-Modell zur fertigen Web-Anwendung. Heidelberg 2000.
Krüger, G.: Handbuch der Java-Programmierung. 3. Aufl., München 2003.

Stahlknecht, P./Hasenkamp, U.: Einführung in die Wirtschaftsinformatik. 8. Aufl., Berlin et al. 1999.
Teubner, R.A.: Organisations- und Informationssystemgestaltung. Theoretische Grundlagen und integrierte Methoden. Wiesbaden 1999.
Wallmüller, E.: Software-Qualitätssicherung in der Praxis. München/Wien 1990.

Die Beantwortung der Fragen erfolgt im WISU-Repetitorium.

Hauptstudium

Unified Modeling Language (UML) — ein bedeutsamer Standard für die konzeptionelle Modellierung

Prof. Dr. Ulrich Frank, Koblenz

Für die Entwicklung betrieblicher Anwendungssysteme sind konzeptionelle Modelle von wesentlicher Bedeutung. Sie zielen darauf ab, die Lücke zwischen der Sichtweise der Anwender und der Software-Entwickler zu schließen. Objektorientierte Modellierungssprachen sind dazu besonders gut geeignet. Seit einiger Zeit gibt es mit der Unified Modeling Language (UML) einen Standard, der für die konzeptionelle Modellierung eine Reihe von vor allem wirtschaftlichen Vorteilen verspricht. Da konzeptionelle Modelle nicht zuletzt darauf gerichtet sind, aktuelle und geplante Abbilder von Unternehmen in den Fachabteilungen zur Diskussion zu stellen, ist der Umgang mit solchen Modellen auch für Wirtschaftswissenschaftler von zunehmender Bedeutung. Nach einer kurzen Einführung in die konzeptionelle Modellierung wird ein Überblick über die UML gegeben, der durch ein anschließendes Beispiel illustriert wird.

1. Bedeutung der konzeptionellen Modellierung

Die Entwicklung von Anwendungsprogrammen, etwa zur Unterstützung der Personalverwaltung, der Finanzbuchhaltung oder des elektronischen Geschäftsverkehrs, erfordert eine sorgfältige **Analyse** der verwendeten Begriffe sowie der benötigten Funktionen und Prozesse. Eine große Herausforderung ist dabei das Erkennen aller relevanten Aspekte. Das gilt in zweifacher Hinsicht. In existierenden Geschäftsprozessen ist hier vor allem an Sonderfälle bzw. Ausnahmen zu denken. Sie werden aus naheliegenden Gründen bei der Erhebung der Anforderungen leicht vergessen. Andererseits ist zu berücksichtigen, dass sich die Anforderungen an Anwendungssysteme im Zeitverlauf ändern können: Geschäftsprozesse werden reorganisiert, neue Arten von Dokumenten sind zu integrieren, neuartige Funktionen abzubilden. Die Systemanalyse sollte solche zukünftigen Entwicklungen antizipieren, um das zu entwerfende System darauf frühzeitig vorzubereiten. Modifikationen zu einem späteren Zeitpunkt sind im Regelfall mit hohen Kosten und erheblichen Risiken verbunden. Um zu leistungsfähigen Generalisierungen zu gelangen, die sowohl Sonderfälle abdecken als auch für zukünftige Entwicklungen offen sind, empfiehlt sich eine intensive Auseinandersetzung mit dem Analysegegenstand. Das wiederum erfordert eine Darstellung, die sich auf die für den Analysezweck wesentlichen Aspekte beschränken und für alle Beteiligte hinreichend anschaulich sein sollte. Bei der Analyse von Anwendungsdomänen empfiehlt sich also die Verwendung geeigneter **Abstraktionen** oder **Modelle**.

Analyse und Entwurf von Systemen sollen Komplexität reduzieren

Systemanalyse ist kein Selbstzweck, sondern dient der Vorbereitung des **Entwurfs** von Software. Der Entwurf selbst zielt auf eine gehaltvolle Vorlage für die **Implementierung**. Er muss deshalb ein präzises Modell des zu realisierenden Systems liefern und gleichzeitig gewissen Randbedingungen der Implementierung Rechnung tragen — etwa den Sprachkonzepten, die von der zu verwendenden Programmier- oder Datenbanksprache

zur Verfügung gestellt werden. Dessen ungeachtet entsteht ein Entwurfsmodell in der Regel nicht allein im Kreis von Software-Entwicklern. Vielmehr sind auch Anwender und Domänenexperten zu beteiligen — es geht ja wesentlich um eine Präzisierung von Begriffen aus der Anwendungswelt. Programmiersprachen oder auch formale Spezifikationssprachen sind für die Entwurfsphase kaum geeignet. Sie sind entweder auf abstrakte Maschinenmodelle oder ein hohes Maß mathematischer Präzision ausgerichtet und sperren sich deshalb gegen Darstellungen, die von allen Beteiligten als verständlich und authentisch empfunden werden. Auf der anderen Seite ist auch die Beschränkung auf eine natürliche Sprache, sei es in Form der Umgangssprache und/oder einer dedizierten Fachsprache, unzureichend: Schließlich sollen Beschreibungen von Anwendungsdomänen Vorgaben für die Implementierung liefern, so dass daraus erwachsende Besonderheiten auch berücksichtigt werden müssen. Gleichzeitig sollten die für den Entwurf verwendeten Konzepte einen engen Bezug zu den Konzepten der Analyse aufweisen, um unnötige Friktionen bei der Verwendung von Analyseergebnissen zu vermeiden.

Konzeptionelle Modelle sollen für die Beteiligten anschaulich sein

Vor dem Hintergrund dieses Spannungsfelds ist die **konzeptionelle Modellierung** entstanden. Die konzeptionelle Modellierung dient einer allen Beteiligten an einem Software-Entwicklungsprozess verständlichen Darstellung einer Anwendungsdomäne. Die Darstellung beschränkt sich dabei auf die für die Erstellung des geplanten Systems wesentlichen Aspekte dieser Domäne. Zusätzlich zu dieser Abstraktion muss ein konzeptionelles Modell auch den Randbedingungen Rechnung tragen, die durch die in einer späteren Phase zu verwendenden Implementierungssprachen entstehen. Dementsprechend sollten konzeptionelle Modelle — in unterschiedlicher Detaillierung und Formalisierung — sowohl für die Analyse als auch für den Entwurf eingesetzt werden.

Die Erstellung konzeptioneller Modelle erfordert geeignete **Modellierungssprachen**. Zu den nach wie vor bedeutendsten Sprachen der konzeptionellen Modellierung gehört das Entity Relationship Model (ERM) mit zahlreichen Derivaten. Auch Datenflussdiagramme (DFD) werden noch häufig verwendet. Seit einigen Jahren werden allerdings zunehmend **objektorientierte Modellierungssprachen** eingesetzt.

> **Frage 1:** Welches sind die wesentlichen Gründe für die Erstellung konzeptioneller Modelle?

2. Objektorientierte Software-Entwicklung

Objektorientierung erfordert spezielle Sprachen für die Modellierung

In den achtziger Jahren wurden objektorientierte Technologien vor allem in Forschungsprojekten und von wenigen innovativen Unternehmen eingesetzt. Mittlerweile haben objektorientierte Programmiersprachen wie C++ und Java eine weite Verbreitung gefunden. Die Verwendung einer objektorientierten Programmiersprache allein ist allerdings nicht hinreichend dafür, die Potenziale objektorientierter Konzepte in sinnvoller Weise zu nutzen. Das erfordert ein angemessenes Verständnis dieser Konzepte sowie den Einsatz einer objektorientierten Modellierungssprache.

2.1. Zentrale Konzepte und Vorteile gegenüber traditionellen Ansätzen

Traditionelle Anwendungssysteme bestehen aus Daten und darauf operierenden Funktionen. In der Kontrollstruktur eines Programms ist festgelegt, in welcher Reihenfolge bzw. bei welchen Ereignissen bestimmte Funktionen aufzurufen sind. Im Unterschied dazu besteht ein objektorientiertes System aus **Objekten**, die über **Nachrichten** kommunizieren.

Objektorientierte Software-Systeme bestehen aus Objekten, die Nachrichten austauschen

Ein Objekt ist ein Artefakt, das einen Gegenstand oder ein Konzept aus der Anwendungsdomäne repräsentiert, also z.B. einen Kunden oder ein Verrechnungskonto. Es kann von seiner Umwelt über seine **Schnittstelle** genutzt werden. Die Schnittstelle besteht aus einer Menge von Diensten, die von den Operationen des Objekts bereitgestellt werden. Auf diese Weise können Eigenschaften eines Objekts (z.B. der Name eines Kunden) abgefragt werden oder Funktionen, die in den Operationen implementiert sind, genutzt werden (z.B. die Berechnung des Alters eines Kunden aus dem in einem Objekt gespeicherten Geburtsdatum).

Die besondere Eignung von Objekten für die Software-Entwicklung liegt darin, dass sie ein vielen Menschen auch außerhalb der Informatik vertrautes Konzept darstellen und zudem ein hohes Abstraktionsniveau unterstützen. Dies gestattet eine Darstellung von Systemen, die nicht nur für Software-Entwickler anschaulich ist. Andererseits wird die Flexibilität von Anwendungen gegenüber zukünftigen Anforderungsänderungen geför-

dert bzw. die Wiederverwendbarkeit existierender Programmteile verbessert. Dazu dienen die folgenden Konzepte.

Objektorientierte Entwicklung erlaubt hohes Abstraktionsniveau

Klassen erlauben die Zusammenfassung gleichartiger Objekte. Dadurch wird es möglich, von den spezifischen Eigenschaften einzelner Objekte (auch **Instanzen** genannt) zu abstrahieren. So legt man z.B. die Eigenschaften der Klasse „Kunde“ mit Hilfe geeigneter Attribute (z.B. „Vorname“, „Nachname“, „Einkommensklasse“) und Operationen (z.B.: „alterInJahren“) fest. Wenn sich nun im Zeitverlauf neue Eigenschaften als sinnvoll erweisen (z.B. das Attribut „E-mail-Adresse“), wird lediglich die Klassenbeschreibung geändert. Damit ist die Erweiterung implizit auch für alle Objekte dieser Klasse wirksam (wobei wir hier von den Besonderheiten einzelner Programmiersprachen und korrespondierender Werkzeuge abstrahieren).

Verkapselung gestattet Abstraktion von interner Struktur des Objekts

Verkapselung bedeutet, dass auf die von einem Objekt verwalteten Daten (die auch als Objekte vorliegen können) von anderen Objekten nicht direkt zugegriffen werden kann. Vielmehr kann nur indirekt über die Schnittstelle eines Objekts zugegriffen werden. In der Außensicht kann man also von der inneren Struktur eines Objekts abstrahieren **(„information hiding“)**. Wenn sich im Zeitverlauf diese Struktur ändert, zieht dies nicht zwangsläufig eine Änderung der Schnittstelle nach sich, was die Wartbarkeit eines Systems erheblich erleichtert.

Generalisierung unterstützt effiziente Systempflege

Durch **Generalisierung** wird es möglich, gemeinsame Eigenschaften einer Menge von Klassen in einer **Oberklasse** zusammenzufassen. Auf diese Weise können Klassenhierarchien gebildet werden. Durch die Bildung von Unterklassen, auch Spezialisierung genannt, können vorhandene Klassen an neue Anforderungen angepasst werden: Eine Unterklasse erbt alle Eigenschaften ihrer Oberklasse und kann mit zusätzlichen Eigenschaften versehen werden, weshalb man in diesem Zusammenhang auch von **Vererbung** spricht. Änderungen eines Systems setzen immer auf dem höchsten sinnvollen Generalisierungsniveau an. Von den jeweiligen Unterklassen kann also abstrahiert werden. Generalisierung kann zu Oberklassen führen, denen in der Anwendungsdomäne keine konkreten Klassen entsprechen. Beispielsweise könnte eine Generalisierung über die Klassen „Rechnung“, „Gutschrift“ und „Lieferschein“ zur Oberklasse „Auftragsabwicklungsdokument“ führen.

Um sicherzustellen, dass in dem zu erstellenden System keine unsinnigen Instanzen gebildet werden, können solche Klassen als **abstrakte Klassen** gekennzeichnet werden. Eine abstrakte Klasse dient lediglich der Nutzung von Generalisierungsmöglichkeiten. Sie hat keine Instanzen.

Neben den bereits genannten Vorteilen verspricht die objektorientierte Software-Entwicklung eine besonders enge Integration der verschiedenen Phasen der Software-Entwicklung: Von der Analyse bis zur Implementierung stellen Objekte das zentrale Konzept dar — wenn auch in unterschiedlicher Detaillierung.

Frage 2: Welche speziellen Vorteile bieten objektorientierte Ansätze für die konzeptionelle Modellierung?

2.2. Vorläufer der UML

Eine objektorientierte Modellierungssprache wird durch eine **abstrakte Syntax**, eine **Semantik** und eine **konkrete Syntax** beschrieben. Die abstrakte Syntax legt fest, wie die Konzepte (wie „Klasse“, „Objekt“, „Attribut“, etc.) einer Sprache verknüpft werden dürfen. Die Semantik legt fest, wie die Konzepte zu interpretieren sind, also z.B. die Bedeutung von Vererbung. Die konkrete Syntax schließlich beschreibt die grafische und gegebenenfalls textuelle Notation. Abstrakte Syntax und Semantik werden häufig mit Hilfe von **Metamodellen** dargestellt (siehe Abschnitt 3.2.).

Objektorientierte Modellierungssprachen sind seit längerem Gegenstand intensiver Forschungsbemühungen. Entsprechend groß ist die Zahl einschlägiger Ansätze.

Objektorientierte Modellierungssprachen in den frühen 90er Jahren

Bis Mitte der neunziger Jahre hatten drei Ansätze besondere Popularität erlangt. Die **„Object Modeling Technique“** (OMT, vgl. Rumbaugh et al. 1991), in einer Arbeitsgruppe von General Electric entwickelt, ist deutlich an traditionellen Sprachen wie dem ERM und Datenflussdiagrammen orientiert. Die Methode von Booch (vgl. Booch 1994) ist im Unterschied dazu durch eine ausgeprägtere Berücksichtigung objektorientierter Konzepte gekennzeichnet. Das in Schweden bei Ericsson entstandene **„Object-Orien-**

ted Software Engineering“ (OOSE, vgl. Jacobson 1993) bietet mit sog. Anwendungsszenarien („use cases“) ein Konzept, das eine durchgängige Beschreibung eines Systems aus der Sicht der Nutzer unterstützt. Während die drei Methoden spezifische Stärken und Schwächen aufweisen, sind sie durch einen gemeinsamen Mangel eingeschränkt: Es fehlt ihnen eine präzise Sprachbeschreibung. Die Sprachkonzepte werden statt dessen durch unvollständige und mehrdeutige Darstellungen sowie ergänzende Beispiele illustriert.

2.3. Gründe für die Standardisierung von Modellierungssprachen

Aus der Sicht von Unternehmen, die Software entwickeln, war die damalige Situation unbefriedigend. Auf der einen Seite hatte sich die Erkenntnis durchgesetzt, dass objektorientierte Ansätze eine wirtschaftlichere Entwicklung und Wartung von Software erlauben, auf der anderen Seite gab es keine einheitliche Modellierungssprache. Vor allem das Bemühen um Investitionsschutz spricht für eine Standardisierung von Modellierungssprachen. Im Einzelnen ist an folgende Aspekte zu denken:

— Der Einsatz einer Modellierungssprache erfordert in der Regel **Modellierungswerkzeuge**, die diese Sprache unterstützen. Bei solchen Werkzeugen handelt es sich um grafische Editoren, die die Erstellung, Verwaltung und Pflege von Modellen unterstützen und zudem die Generierung von Code erlauben. Die Einführung solcher Werkzeuge ist mit hohen Kosten verbunden. Das gilt vordergründig für den Kauf und die Wartung, vor allem aber für die Schulung der Mitarbeiter. Wenn die gewählte Modellierungssprache im Zeitverlauf durch andere verdrängt wird, sind diese Investitionen gefährdet: Der Anbieter des entsprechenden Modellierungswerkzeugs sieht keine hinreichenden Absatzchancen mehr und stellt das Produkt deshalb ein. Ein Standard hingegen verspricht größere Unabhängigkeit von einem Anbieter.

Standardisierung verspricht besseren Investitionsschutz

— Die Erstellung konzeptioneller Modelle erfordert hohen Aufwand. Ein Standard hilft nicht nur, die in die Modelle getätigten Investitionen zu schützen, sondern vereinfacht auch die **Wiederverwendung von Modellen** in anderen Kontexten. Auf diese Weise wird auch ein Marktplatz für konzeptionelle Modelle möglich. In letzter Zeit hat diese Vorstellung unter dem Etikett **„Referenzmodell“** erhebliche Resonanz gefunden.
— Wenn eine standardisierte Modellierungssprache verwendet wird, vergrößert sich die Chance, am **Arbeitsmarkt** einschlägig qualifizierte Mitarbeiter zu finden. In Zeiten eines ohnehin gravierenden Mangels an DV-Fachkräften ist dies ein gewichtiges Argument.

Frage 3: Warum ist es sinnvoll, eine Modellierungssprache zu standardisieren?

3. Unified Modeling Language (UML)

Die Entstehung der UML geht auf zwei parallele Entwicklungen zurück. Zunächst einigten sich Rumbaugh und Booch darauf, ihre rudimentären Sprachentwürfe zu einer neuen, gemeinsamen Modellierungssprache weiter zu entwickeln und zusammen mit zugehörigen Werkzeugen zu vermarkten. Die neue Sprache wurde Unified Modeling Language genannt. Kurze Zeit später schloss sich auch Jacobson der Firma Rational an. Im Juni 1996 veröffentlichte die Object Management Group (OMG), ein Konsortium aus Anbietern und Anwendern von Informationstechnologie, einen „Request for Proposals“, in dem Vorschläge für standardisierungsfähige Sprachen eingefordert wurden.

Die UML ist ein Produkt von drei Fachautoren und der OMG

Bis Januar 1997 sind sechs Vorschläge bei der OMG eingegangen. Kurze Zeit später einigten sich die Einsender unter Mitwirkung der OMG darauf, gemeinsam einen Vorschlag weiter zu verfolgen, der wesentlich auf der UML der amerikanischen Firma Rational basierte. Nach einer Reihe von Überarbeitungen wurde die UML 1998 von der OMG als Standard verabschiedet. Mittlerweile liegt die Version 1.3 vor. Man kann davon ausgehen, dass die UML in der kommerziellen Software-Entwicklung heute die wichtigste objektorientierte Modellierungssprache ist.

Frage 4: Wer steht hinter der UML?

3.1. Diagrammarten und Erweiterungsmöglichkeiten

Konzeptionelle Modelle sind Abstraktionen. Im Mittelpunkt der objektorientierten Modellierung stehen **Klassendiagramme**. Sie dienen der Beschreibung statischer Systemaspekte. Ein Klassendiagramm besteht aus Klassen und Beziehungen zwischen Klassen.

Eine Klasse wird als Rechteck dargestellt. Neben dem Klassennamen kann das Rechteck auch die Namen von Attributen und Operationen enthalten.

Klassendiagramme stellen die statische Struktur eines Systems dar

Generalisierungsbeziehungen werden als Pfeil dargestellt, wobei die Pfeilspitze auf die Oberklasse zeigt. In Abb. 1 ist „Konto“ auf diese Weise als Oberklasse von „Girokonto“ und „Sparkonto“ gekennzeichnet. Zudem ist „Konto“ als abstrakte Klasse ausgewiesen. Assoziationen zwischen Klassen werden durch Linien zwischen jeweils zwei Klassensymbolen repräsentiert. An die Linien werden **Kardinalitäten** angetragen, die die minimale und maximale Anzahl von den an der Assoziation beteiligten Objekten einer Klasse beschreiben. So drückt etwa die Kardinalität 0..* in der Assoziation zwischen Bank und Kontoinhaber aus, dass eine Bank minimal kein Konto führt, maximal beliebig viele. Außerdem kann eine Assoziation mit einem Namen gekennzeichnet werden, für den mittels eines Pfeils eine Leserichtung angegeben wird. Attribute werden durch einen Namen und eine Klasse spezifiziert. Die Klasse eines Attributs beschreibt seine Bedeutung. So wird im Beispiel das Attribut „Geburtsdatum“ durch die Klasse „Date“ charakterisiert. Operationen können durch die jeweils benötigen Parameter näher beschrieben werden.

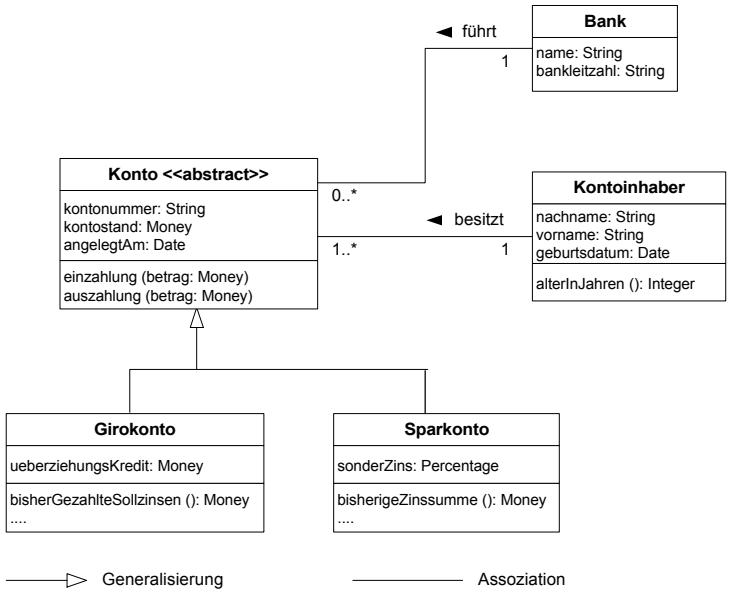


Abb. 1: Klassendiagramm in UML

In Ergänzung zu Klassendiagrammen können in Abhängigkeit vom jeweils vorherrschenden Blickwinkel weitere Systemaspekte von Bedeutung sein, z.B. der Nachrichtenaustausch zwischen Objekten oder zulässige Zustandsänderungen von Objekten.

Jede UML-Diagrammart unterstützt eine bestimmte Sicht

Die UML bietet denn auch eine Reihe unterschiedlicher Diagrammarten, die jeweils für eine bestimmte Abstraktion gedacht sind. Insgesamt erlaubt die UML zur Zeit acht **verschiedene Diagrammarten**, die in der folgenden Übersicht dargestellt sind.

Abstraktion	Diagrammart	Zentrale Konzepte
Statische Sicht	Class Diagram	Class, Association, Generalisation, Attribute, …
Systemnutzung	Use Case Diagram	Use Case, Actor, Association
Implementierungssicht	Component Diagram	Component, Interface, Dependency
Einsatzsicht	Deployment Diagram	Node, Component, Dependency
Dynamische Sicht	State Chart Diagram	State, Event, Transition, Action
Prozesssicht	Activity Diagram	State, Activity, Completion Transition, Fork, Join
Interaktionssicht	Sequence Diagram	Interaction, Message, Activation
	Collaboration Diagram	Collaboration, Interaction, Message

Tab. 1: Übersicht der Diagrammarten

Jede Diagrammart ist mit einer bestimmten Sicht bzw. Abstraktion verknüpft. An dieser Stelle ist eine ausführliche Beschreibung aller Diagrammarten (eine solche findet sich in Rumbaugh et al. 1999) nicht möglich. Statt dessen skizzieren wir die mit den Diagramm-

arten verbundenen Abstraktionen durch die Fragen, mit denen sich der Modellierer bzw. Betrachter jeweils dem Modellierungsgegenstand nähert:

Auswahl der Diagrammart nach Entwurfszweck

- Welche Klassen sind für die Abbildung des Gegenstandsbereichs geeignet? Welche Attribute und Operationen weisen die Klassen auf? Welche Beziehungen gibt es zwischen den Klassen bzw. Objekten? **(Class Diagram)**
- Welche Fälle der Systemnutzung gibt es? Wie stellen sich in jedem Fall die Anforderungen an die Systemfunktionalität aus der Sicht des Akteurs dar? Welche Dienste erwartet der Akteur von den verfügbaren Objekten? **(Use Case Diagram)**
- Welche Komponenten (vorhandene Anwendungen, implementierte Klassen und Klassengruppierungen) sind für die Systemarchitektur wesentlich? Welche Beziehungen/Abhängigkeiten gibt es zwischen den Komponenten? Welche Schnittstellen bieten die Komponenten und das System? **(Component Diagram)**
- Welche Systemressourcen (Computer, Massenspeicher, periphere Geräte) werden von den Komponenten benötigt? **(Deployment Diagram)**
- Welche wesentlichen Ereignisse sind für Objekte einer Klasse zu beachten? Wie sollen die Objekte dieser Klasse auf die Ereignisse reagieren? In welcher Weise ändert sich beim Auftreten eines Ereignisses gegebenenfalls der Zustand eines Objekts? **(State Chart Diagram)**
- Welche Prozesse sind für das zu erstellende System von Bedeutung? Wie lassen sich diese Prozesse als zeitliche Folge von Aktivitäten beschreiben? **(Activity Diagram)**
- Welche Nachrichten schicken sich Objekte verschiedener Klassen zu? In welcher zeitlichen Reihenfolge erfolgt der Nachrichtenaustausch? **(Sequence Diagram)**
- Wie können Objekte, die untereinander Nachrichten austauschen (die also „zusammenarbeiten"), sich gegenseitig identifizieren? **(Collaboration Diagram)**

Für die Systementwicklung ist es von zentraler Bedeutung, dass die in verschiedenen Diagrammen dargestellten Sichten auf ein System in konsistenter Weise integriert werden. Integration erfordert gemeinsame Konzepte. In Kap. 4 werden ausgewählte Diagramm-arten sowie deren Integration anhand eines Beispiels näher erläutert.

UML erlaubt individuelle Erweiterungen

Auch wenn die UML eine Fülle von Diagrammarten bietet, mag es Modellierungsaufgaben geben, für die kein zufriedenstellendes Konzept verfügbar ist. Für solche Fälle erlaubt die UML individuelle Erweiterungen. Ein wichtiges Instrument für solche Erweiterungen stellen **„Stereotypes"** dar. Ein Stereotype wird eingeführt, indem man ein existierendes Sprachkonzept mit einer speziellen Interpretation (also einer individuellen Semantik) versieht.

Zudem kann man dieses Konzept durch ein selbst definiertes grafisches Symbol kennzeichnen. Auf diese Weise könnte man etwa für die Organisationsmodellierung das Konzept „Organisationseinheit" einführen, indem man das vorhandene Konzept „Klasse" um eine entsprechende Semantik ergänzt. Dazu würde etwa gehören, dass eine Organisationseinheit anderen Organisationseinheiten übergeordnet sein kann, aber nicht sich selbst. Der zusätzlichen Flexibilität, die die UML durch Stereotypes gewinnt, steht ein gravierender Nachteil gegenüber: Die für Stereotypes festgelegte Semantik ist nicht Bestandteil des Standards, so dass deren Nutzung die Wiederverwendung von damit beschriebenen Modellen erheblich einschränkt.

Frage 5: Warum sind verschiedene Diagrammarten erforderlich?

3.2. Sprachspezifikation

Die UML ist eine komplexe Sprache. Dementsprechend umfangreich ist die Sprachbeschreibung.

Sprachbeschreibung erfordert eine Metasprache

Sie besteht aus einer **Sprachspezifikation** und einem ergänzenden Handbuch, das die Anwendung der Sprachkonzepte beschreibt. Die Sprachspezifikation besteht aus einem **Metamodell** und ergänzenden Integritätsbedingungen. Ein Metamodell ist gleichsam das Modell einer Sprache. Die aktuelle Spezifikation der UML findet sich unter www.rational.com bzw. unter www.omg.org. Abb. 2 auf der folgenden Seite illustriert die verschiedenen Sprachebenen, die bei der Modellierung eines Realitätsausschnitts zu beachten sind.

Im Fall der UML wird das Metamodell mit Hilfe eines vereinfachten Klassendiagramms dargestellt. Ein konkretes UML-Diagramm ist eine Instanz des UML-Metamodells. Grundsätzlich könnte die Sprachspezifikation auch mittels einer Grammatik erfolgen. Die

Verwendung von Metamodellen hat allerdings zwei Vorteile: Durch die Verwendung gleicher Darstellungskonzepte für die Sprachbeschreibung und die Sprachanwendung wird ein Paradigmenbruch vermieden, der den Sprachanwendern einen leichteren Zugang zur Sprachspezifikation eröffnen sollte.

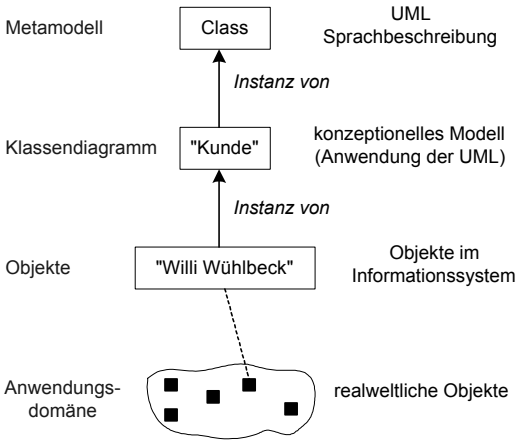


Abb. 2: Ebenen der Modellierung

Die Anwendung einer Modellierungssprache empfiehlt geeignete Modellierungswerkzeuge. Für die Realisierung solcher Werkzeuge ist ein objektorientierter Ansatz besonders gut geeignet. Metamodelle bieten eine gute Grundlage für den Entwurf von Werkzeugen, da sie leicht in Klassendiagramme überführt werden können.

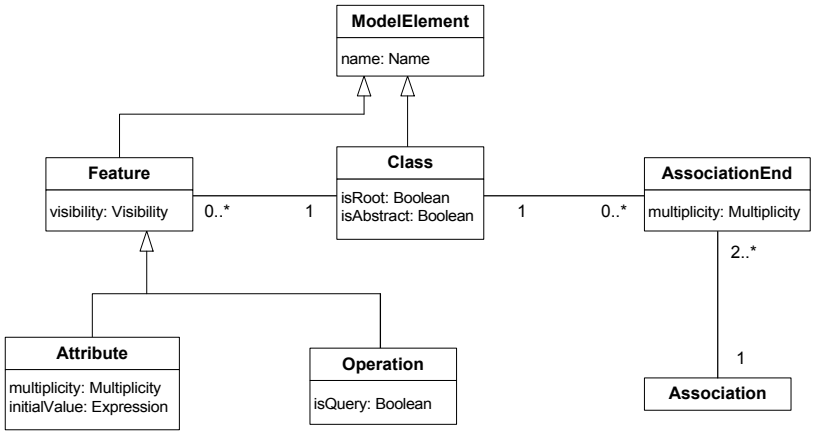


Abb. 3: Vereinfachter Ausschnitt des Metamodells der UML

Darstellung von Assoziationen

Abb. 3 zeigt einen Ausschnitt aus dem Metamodell der UML. Er stellt die Sprachkonzepte dar, die im Klassendiagramm in Abb. 1 angewendet werden. Eine Klasse (eine Instanz von „Class“) kann beliebig viele Eigenschaften („Features“) haben, bei denen es sich jeweils entweder um Attribute oder Operationen handelt. Etwas schwieriger sind die Konzepte nachzuvollziehen, die für die Darstellung von **Assoziationen** verwendet werden. Die Assoziation zwischen „Konto“ und „Kontoinhaber“ in Abb. 1 wird wie folgt konstruiert: Die Klasse „Konto“ ist eine Instanz von „Class“, die mit einer Instanz von „AssociationEnd“ verknüpft ist. Dieser Instanz ist die Kardinalität („multiplicity“) 1..* zugeordnet. Außerdem ist sie mit einer Instanz von „Association“ verknüpft, die wiederum mit einer zweiten Instanz von „AssociationEnd“ (der die Kardinalität 1 zugeordnet ist) verbunden ist. Diese Instanz von „AssociationEnd“ wiederum ist mit einer Instanz von „Class“, nämlich der Klasse „Kontoinhaber“ verknüpft.

Aus Sicht der Informatik ist es wünschenswert, dass ein Metamodell ein korrektes Modell der jeweiligen Sprache liefert. Die Sprachbeschreibung sollte die Identifikation unzulässiger Modelle gewährleisten und gleichzeitig erlauben, die Menge aller zulässigen Modelle zu generieren. Die Sprachbeschreibung sollte zudem vollständig sein, d.h., alle in der Sprache verwendeten Konzepte sowie die Bedingungen ihrer Verwendung sollten definiert sein. Diese Forderungen erfüllt die UML nur eingeschränkt, da sie eine Reihe **semantischer Lücken** enthält.

UML enthält eine Reihe von Mehrdeutigkeiten

Einige Konzepte, unter anderem auch das wichtige Konzept „Vererbung“, sind nicht präzise definiert. Ihre Verwendung erfordert also eine entsprechende Interpretation des Be-

nutzers. Beim Austausch oder der Wiederverwendung von Modellen sind deshalb Konventionen festzulegen, um die in UML enthaltenen Mehrdeutigkeiten zu eliminieren. Ergänzend zu den offiziellen Dokumenten der OMG gibt es eine Reihe von Lehrbüchern, die in den Umgang mit der UML einführen (vgl. Fowler/Scott 1997; Oestereich 1998; Page-Jones 1999). Eine kritische Bewertung der UML findet sich in Frank/Prasse 1997.

4. Ein Beispiel

Um die Beziehungen zwischen verschiedenen Diagrammen der UML zu verdeutlichen, betrachten wir im Folgenden ein einfaches Beispiel. In einem Unternehmen soll eine neue Software für die Auftragsbearbeitung erstellt werden. Zunächst wird untersucht, welche Anforderungen die zukünftigen Nutzer des Systems haben. Dazu wird für jeden Benutzer ermittelt, welche Anwendungsszenarien, also Anlässe zur Systemnutzung, es jeweils gibt. Jedes Anwendungsszenario ist dadurch gekennzeichnet, dass ein Benutzer vom System bestimmte Informationen nachfragt und/oder die Ausführung bestimmter Funktionen erwartet. Eine solche Erhebung kann mit Hilfe von Use Cases durchgeführt werden.

Anwendungsszenarien stellen typische Nutzungsformen dar

Abb. 4 veranschaulicht die grafische Darstellung des Anwendungsfalls **„Auftragsbearbeitung“**, in den weitere Anwendungsfälle eingebettet sind. Da die Grafik allein wenig aussagt, ist die Dokumentation unbedingt durch ausführliche natürlichsprachliche Beschreibungen der Anwendungsfälle zu ergänzen.

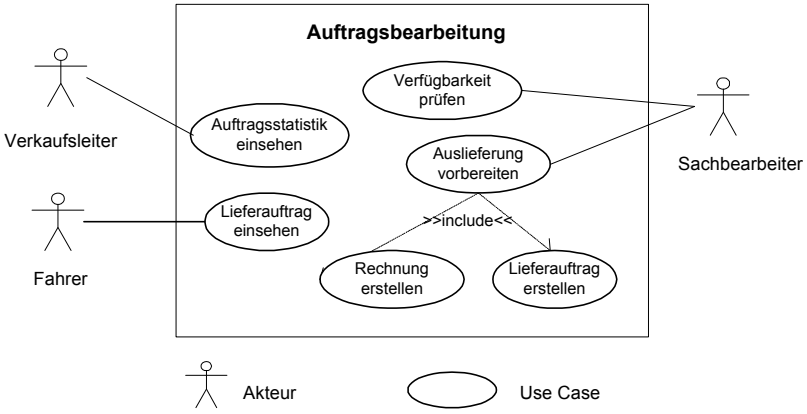


Abb. 4: Darstellung der Auftragsbearbeitung als Anwendungsfall (Use Case)

Zunächst Funktionen bzw. Prozesse ermitteln

Der objektorientierte Entwurf der Software macht es erforderlich, geeignete Objekte bzw. Klassen zu identifizieren. Es hat sich allerdings als wenig erfolgversprechend erwiesen, unmittelbar nach den Objekten in der Anwendungsdomäne zu fragen. Statt dessen ist es sinnvoller, zunächst die Aufgaben bzw. Prozesse zu betrachten: Die meisten Menschen sind eher in der Lage, ihr Arbeitsumfeld mit Hilfe von Prozessen zu beschreiben.

Abb. 5 auf der folgenden Seite zeigt ein **Acitivity Diagram**, in dem einzelne Aufgaben (Aktivitäten) zeitlich geordnet sind (von oben nach unten). Zudem können alternative Abläufe durch das Einfügen von Bedingungen (durch eine Raute veranschaulicht) dargestellt werden. Für jede Aktivität kann nun ermittelt werden, welche Objekte (bzw. deren Klassen) benötigt werden. Um die Klassen zu spezifizieren, ist darüber hinaus nach den erforderlichen Operationen und Attributen zu fragen.

Wie die für die Implementierung nötige Integration von Aktivitäts- und Klassendiagrammen durchzuführen ist, wird durch die UML nicht genau spezifiziert. Aktivitäten können als Operationen interpretiert werden, so dass beide Diagrammarten über diese gemeinsamen Bestandteile verknüpft werden könnten. Im vorliegenden Beispiel werden Aktivitäten als Aufgaben im Rahmen eines Geschäftsprozesses interpretiert. Die Ausführung dieser Aufgaben erfordert unter Umständen den Rückgriff auf Objekte in einem Informationssystem. Die Verwendung von Objekten/Klassen in einzelnen Aktivitäten ist in Abb. 5 durch gestrichelte Linien visualisiert.

Die Beispiele basieren auf stark vereinfachenden Annahmen. So sind Kunden häufig nicht nur Personen, sondern auch Organisationen. Ein Auftrag kann auf mehrere Rechnungen verteilt werden etc. Gerade wegen seiner Unzulänglichkeiten verdeutlicht das Beispiel, wie wichtig es ist, die in einem Modell implizit enthaltenen Annahmen sorgfältig daraufhin zu prüfen, ob sie allen Einzelfällen im betrachteten Realitätsausschnitt gerecht werden.

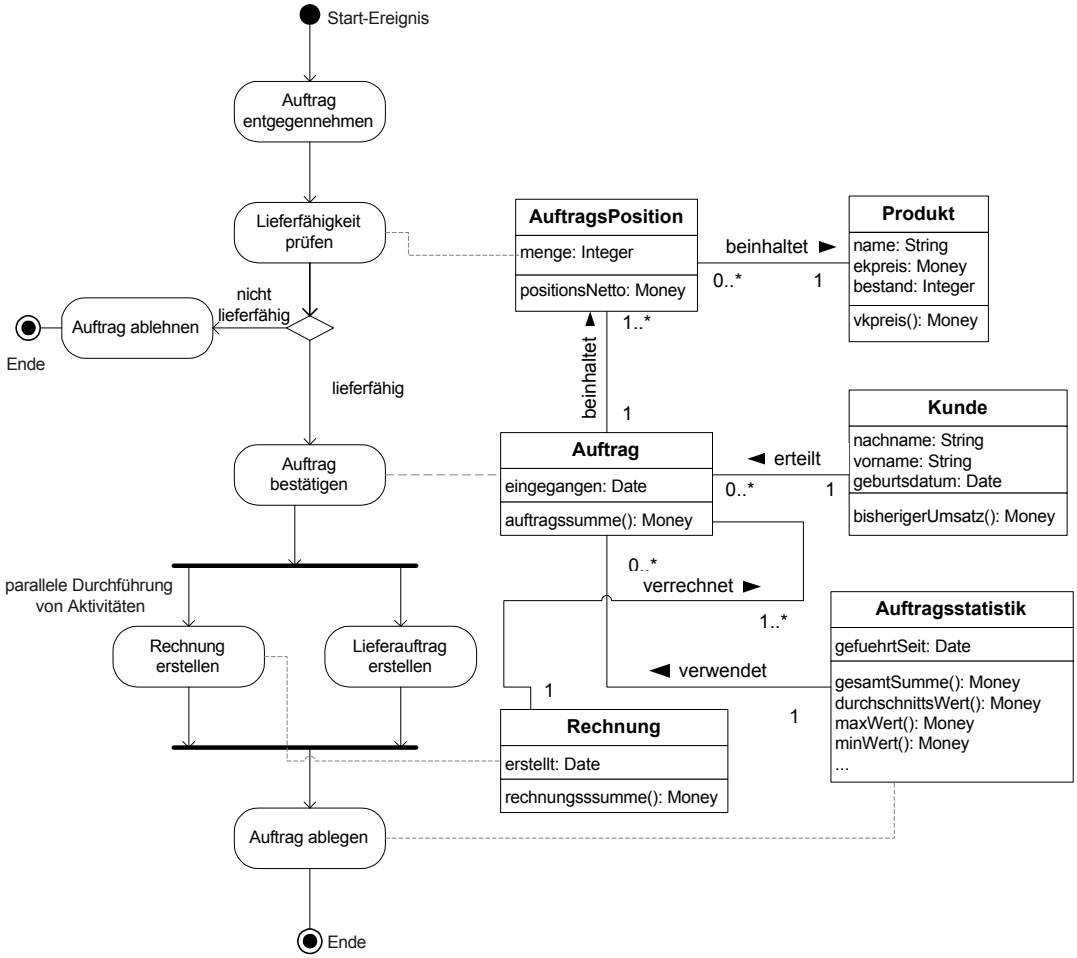


Abb. 5: Aktivitätsdiagramm und zugehöriges Klassendiagramm

5. Abschließende Bemerkungen

UML hat auch für Ökonomen Bedeutung

Konzeptionelle Modelle stellen ein Medium für die Kommunikation zwischen Software-Entwicklern, Auftraggebern und Anwendern dar. Die UML ist deshalb nicht nur eine Sprache für Informatiker oder Wirtschaftsinformatiker, sondern auch für Wirtschaftswissenschaftler.

Dabei ist vor allem an solche Diagrammarten zu denken, die einen offenkundigen Bezug zur Anwendungsdomäne haben und weniger mit software-technischen Details behaftet sind. Dazu gehören in erster Linie die in dem dargestellten Beispiel verwendeten Diagrammarten. Auch wenn die Standardisierung einer Sprache für die konzeptionelle Modellierung erhebliche Vorteile verspricht (siehe Abschnitt 3.2.), reicht ein Sprachstandard allein nicht aus, um Software-Entwicklungsprojekte erfolgreich durchzuführen. Vielmehr ist es erforderlich, entsprechende Projekte sorgfältig zu planen und zu überwachen. Die OMG hat bewusst darauf verzichtet, einen Prozess für die Anwendung der UML zu standardisieren, da die Vielfalt denkbarer Software-Entwicklungsprojekte allenfalls sehr abstrakte und deshalb im Einzelfall wenig hilfreiche Vorgaben ermöglichen würde. Es gibt allerdings mittlerweile eine Reihe von Lehrbüchern, die darauf zielen, diese Lücke zu füllen (vgl. Kruchten 1999; Jacobson et al. 1999). Sie geben Empfehlungen für die Vorgehensweise, Heuristiken und Beispiele.

Aus der Sicht der Wirtschaftsinformatik ist die UML trotz der Fülle der angebotenen Diagrammarten nicht vollständig. Insbesondere fehlen Modellierungssprachen, die eine detaillierte Darstellung von Geschäftsprozessen oder strategischer Zusammenhänge (z.B. in Form von Wertketten) ermöglichen. Die UML enthält keine speziellen betriebswirtschaftlichen Konzepte. Sie ist deshalb für die Unternehmensmodellierung nur begrenzt geeignet. Neben eigenständigen Ansätzen zur Unternehmensmodellierung aus der Wirtschaftsinformatik, gibt es seit einiger Zeit einschlägige Forschungsansätze, die auf eine entsprechende Erweiterung der UML um sog. „Profiles“ — spezielle Modellierungssprachen auf der Basis der UML — gerichtet sind. Es bleibt abzuwarten, ob solche Erweiterungen eines Tages in die UML aufgenommen werden.

Literaturempfehlungen:

Booch, G.: Object-oriented Analysis and Design with applications. Redwood City, Ca. 1994.
Fowler, M./Scott, K.: UML Destilled: Applying the Standard Object Modeling Language. Reading, Mass. et al. 1997.
Jacobson, I.: Object-Oriented Software Engineering. Reading, Mass. et al. 1993.
Jacobson, I./Booch, G./Rumbaugh, J.: Unified Software Development Process. Reading, Mass. et al. 1999.
Frank, U./Prasse, M.: Ein Bezugsrahmen zur Beurteilung objektorientierter Modellierungssprachen — veranschaulicht am Beispiel von OML und UML. Arbeitsberichte des Institut für Wirtschaftsinformatik der Universität Koblenz-Landau, Nr. 7, 1997.
Kruchten, P.: Der Rational Unified Process: Eine Einführung. München 1999.
Oestereich, B.: Objektorientierte Softwareentwicklung. München/Wien 1998.
Page-Jones, M.: Fundamentals of Object-Oriented Design in UML. Reading, Mass. et al. 1999.
OMG (ed.): OMG Unified Modeling Language Specification. Version 1.3 1999, digitales Dokument, verfügbar unter http://www.rational.com.
Rumbaugh, J. et al.: Object-oriented Modeling and Design. Englewood Cliffs 1991.
Rumbaugh, J./Jacobson, I./Booch, G.: The Unified Modeling Language Reference Manual. Reading, Mass. et al. 1999.

Die Beantwortung der Fragen erfolgt im WISU-Repetitorium.

Die Examensklausur aus der Wirtschaftsinformatik

Die folgenden Aufgaben wurden im Prüfungstermin Sommersemester 1999 von Prof. Dr. Dieter Bartmann an der Universität Regensburg in der Diplomprüfung für die Studiengänge Betriebswirtschaftslehre und Wirtschaftsinformatik im Fach „Bankinformatik und Integrierte Systeme" gestellt. Die gesamte Prüfung besteht aus fünf Aufgabenblöcken à 60 Punkte, die sich aus einem Pflichtteil (40 Punkte) und einem Wahlteil (20 Punkte) zusammensetzen. Für die gesamte Klausur standen 300 Minuten Bearbeitungszeit zur Verfügung. Dabei entsprach ein erzielbarer Punkt einer Minute Bearbeitungszeit. Wir geben hier eine Auswahl aus den Aufgaben wieder.

Thema: Aufgabenblock Integrierte Systeme

Aufgabe: CAQ und Qualitätsregelkreise (20 Punkte)

a) Skizzieren Sie das System der Qualitätsregelkreise (5 Punkte).
b) Welche Rolle spielt dabei die Betriebsdatenerfassung (BDE)? Priorisieren Sie die in den jeweiligen Regelkreisen ermittelbaren Qualitätsmängel hinsichtlich ihrer Kostenrelevanz für den Fertigungsbetrieb. Nennen Sie für jeden Regelkreis einige Beispiele für auftretende Kosten. Denken Sie dabei an die Mängelursachen und den erforderlichen Mängelbehebungsaufwand (15 Punkte).

Aufgabenblock IT und Strategische Führung

Aufgabe: Virtuelle Organisation und IT-Unterstützung (20 Punkte)

a) IT ist ein bestimmender Faktor für das Konzept der virtuellen Organisation. Beschreiben Sie die virtuelle Organisation in der institutionellen Sicht mit ihren konstituierenden Charakteristika sowie deren potentielle Problemfelder (12 Punkte).
b) Können Systeme aus dem Bereich Computer Supported Cooperative Work (CSCW) die Gestaltung virtueller Organisationen unterstützen? Definieren Sie zur Beantwortung der Frage kurz CSCW und grenzen Sie

mationen ein Prozess, der die Schritte **Vorbereitung, Definition, Überprüfung und Modifikation** umfasst. Die Überprüfung bezieht sich dabei (1) auf die syntaktische Gültigkeit („Entspricht das Matching der zu Grunde liegenden Transformationssprache?“), (2) die Vollständigkeit („Werden alle Datenelemente des Quell- und Zielformats berücksichtigt?“) und (3) die Korrektheit („Sind die Transformationen inhaltlich korrekt?“).

Literaturempfehlungen:

Ariba, Inc.: cXML 1.2.009. Online: http://xml.cxml.org (Stand: 17.01.2004).

Beul, M./Bittscheidt, C./Leukel, J./Spies, T.: Behandlung von Informationsdefiziten und -verlusten bei der Transformation von XML-Geschäftsdaten. In: Proceedings der 5. Paderborner Frühjahrstagung „Innovationen im E-Business“, Paderborn, 2003, S. 159 - 168.

Dorloff, F.-D./Leukel, J./Schmitz, V.: Datentransformation bei XML-basierten Geschäftsdokumenten. In: WISU , 33. Jg. (2004), S. 87 - 94.

Leukel, J./Schmitz, V./Dorloff, F.-D.: Coordination and Exchange of XML Catalog Data in B2B. In: Proceedings of the 5th International Conference on Electronic Commerce Research (ICECR-5), Montreal 2002.

Rahm, E./Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. In: The VLDB Journal, Vol. 10 (2001), S. 334 - 350.

Schmitz, V./Kelkar, O./Pastoors, T.: Spezifikation BMEcat Version 1.2, 2001. Online: http://www.bmecat.org (Anmeldung erforderlich, Stand: 17.01.2004).

Schmitz, V./Leukel, J. Dorloff, F.-D.: Does B2B Data Exchange Tap the Full Potential of XML Schema Languages. In: Proceedings of the 16th Bled Electronic Commerce Conference, Bled 2003, S. 172 - 182.

Wüstner, E./Hotzel, T./Buxmann, P.: Converting Business Documents: A Classification of Problems and Solutions using XML/XSLT. In: Proceedings of the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems (WECWIS 2002), Newport Beach 2002, S. 61 - 68.

Die Fragen werden im WISU-Repetitorium beantwortet.

Hauptstudium

Model Driven Architecture

Prof. Dr. Rainer Thome / Dipl.-Kfm. Martin Böhn /
Dipl.-Kfm. Alexander Hagn, Würzburg

Software-Entwickler und System-Designer müssen bei größeren Änderungen oder Erweiterungen vorhandener Systeme oft neue Entwürfe anfertigen, da sich die alten Konzepte nicht wiederverwenden lassen. „Model Driven Architecture“ hilft, Modelle bis hin zu fertigen Programmen und komplexen Architekturen zu realisieren.

1. Grundlagen

Die Model Driven Architecture ist ein Konzept, das von der Object Management Group (OMG) entwickelt und standardisiert wurde. Die OMG ist ein Verbund von über 800 Unternehmen, der die Festlegung herstellerneutraler Standards zur Interoperabilität und Portabilität von Software zum Ziel hat.

1.1. Zielsetzung

Mit Hilfe der Model Driven Architecture wird die Durchgängigkeit der Entwicklung und Einführung von Anwendungssystemen von den geplanten Konzepten und Modellen bis zur automatisierten Generierung von fertigem Programm-Code angestrebt. Dies ist nur erreichbar, wenn die zu Grunde liegenden Modelle stark abstrahiert und eindeutig formalisiert wurden. Darüber hinaus wird durch die verschiedenen Abstraktionsgrade eine hohe **Wiederverwendbarkeit** der erstellten Software-Designs für künftige Änderungen und Erweiterungen bzw. Produktvarianten erreicht. Dazu müssen die in der Systemarchitektur spezifizierten Funktionen und Schnittstellenbeschreibungen von Details der später

eingesetzten Plattformen und Systeme getrennt werden. Der Ansatz der Model Driven Architecture umfasst ein mehrstufiges Modell und schließt eine Reihe von frei zugänglichen und zum Teil standardisierten Technologien ein, um diese Ziele zu erreichen (vgl. Roßbach/Stahl/Neuhaus 2003, S. 22 - 24).

1.2. Stufenmodell

Innerhalb der Model Driven Architecture erfolgt eine Trennung von Facharchitektur, systemunabhängiger Software-Architektur und systemspezifischer Anwendungsarchitektur. Ergänzt wird diese Schrittfolge durch die Spezifikation der technischen Plattform-Architektur.

Computer Independent Model (CIM)

Zu Beginn eines Entwicklungsprozesses sind die fachlichen Anforderungen an die künftigen Anwendungssysteme in ein computerunabhängiges Modell zu überführen. Im CIM erfolgt die Modellierung der unternehmenseigenen **Facharchitektur** durch die zuständigen Experten unabhängig von einer späteren informationstechnologischen Unterstützung. Geschäftsabläufe sowie die beteiligten Mitarbeiter werden erfasst und die Teilaufgaben, Bedingungen und Schnittstellen festgelegt. Zur Darstellung eignen sich Use-Case- oder Aktivitätsdiagramme der UML.

Platform Independent Model (PIM)

Die Struktur des Systems, die zu verwendenden Komponenten sowie deren Funktionalitäten und Interaktionen werden im PIM beschrieben. Zur Gewährleistung eines hohen Wiederverwendbarkeitsgrades ist es notwendig, eine allgemeine, von der konkreten technischen Umsetzung unabhängige Architektur zu formulieren. Das PIM spezifiziert somit lediglich das geforderte Fachkonzept der künftigen Anforderung. Dieses **plattformunabhängige Modell** kann zur Erstellung verschiedener plattformspezifischer Modelle herangezogen werden. Im Idealfall werden Entwicklung und spätere Anpassung nur im PIM vorgenommen, welches automatisch in PSM und weiter bis hin zum ausführbaren Code überführt werden soll (vgl. Bohlen/Starke 2003, S. 52).

Platform Specific Model (PSM)

Auf dieser Grundlage wird die Umsetzung des PIM auf eine geeignete Systemplattform konkretisiert. Dadurch ergibt sich ein **plattformabhängiges Modell**, das der Spezifikation der zu verwendenden Technologien, beispielsweise EJB oder .NET, dient. Es soll möglichst automatisch aus dem PIM abgeleitet werden. Hierzu existieren bereits standardisierte Transformationsregeln für einige Plattformen. Die gestützte Überführung von PIMs in PSMs für weitere Betriebssysteme und Programmiersprachen ist in Planung.

In Abhängigkeit von der zu Grunde liegenden Systemplattform werden verschiedene PSM unterschieden, beispielsweise EJB-, CCM- oder .NET-Modelle. Über Profile kann ein PSM in Code überführt werden. Jedes technologiespezifische PSM ist dabei einer entsprechenden Abbildungsvorschrift zugeordnet (vgl. OMG MDA Guide 2003, S. 15 - 17).

Enterprise Deployment Model (EDM)

Wird schließlich das PSM um die betriebliche IT-Infrastruktur erweitert, erhält man das **systemspezifische Modell**. Beim Übergang der Modelle zwischen den einzelnen Stufen sind exakte Transformationsregeln notwenig, um einen hohen Automationsgrad und eine geringe Fehlerquote zu gewährleisten. Im EDM werden beispielsweise Komponenten-Container, Applikationsserver oder Anforderungen an die spätere Einführung und Verteilung der Anwendung festgelegt. Das EDM ist nicht Bestandteil der von der OMG verabschiedeten Spezifikationen und stellt eine Erweiterung dar (vgl. Andresen 2003, S. 70 - 71). Abb. 1 fasst das mehrstufige Konzept der Model Driven Architecture zusammen.

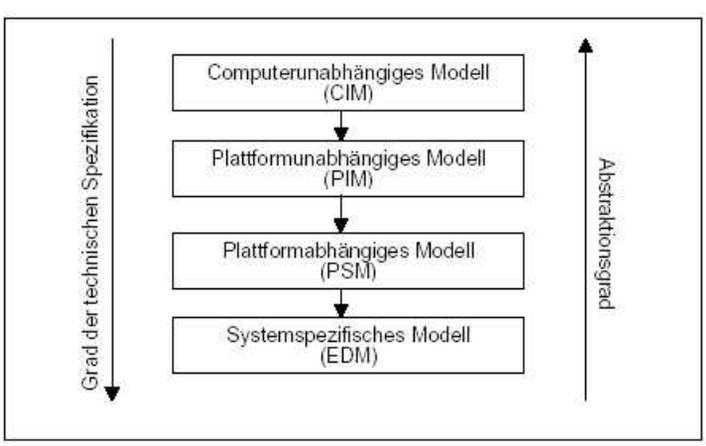


Abb. 1: Aufbau der Model Driven Architecture

1.3. Technologien

Verschiedene Technologien können als Plattform dienen

Die der Model Driven Architecture zu Grunde liegenden Technologien und Spezifikationen bestehen in weit verbreiteten und zum Teil standardisierten Ansätzen der System- und Software-Entwicklung. Als übergeordnete Metamodellierungssprache steht dem Anwendungsentwickler die Meta Object Facility (MOF) zur Verfügung, mit deren Hilfe die Verwendung der Unified Modeling Language (UML) und des Common Warehouse Model (CWM) formalisiert werden. Diese drei **Modellierungsansätze** wurden ebenfalls von der OMG standardisiert. Auf Basis der formulierten Modelle können verschiedene Technologien als **Plattformen** genutzt werden. Eine genaue Definition des Begriffs Plattform wurde von der OMG allerdings bisher nicht verabschiedet. Darunter können einerseits Programmiersprachen wie beispielsweise Java oder C++ sowie andererseits Middleware-Systeme subsumiert werden. Zu den wichtigsten Middleware-Plattformen zählen insbesondere die stark verbreiteten Ansätze Java 2 Enterprise Edition (J2EE) von Sun Microsystems als auch .NET von Microsoft. Als weitere mögliche Ansätze stehen CORBA, XML/XMI sowie Web Services zur Verfügung (vgl. Roßbach/Stahl/Neuhaus 2003, S. 22 - 24).

1.4. Transformation

Transformationsregeln sind notwendig

Für die Überführung der Modelle auf den einzelnen Ebenen des Stufenmodells der Model Driven Architecture sind besondere **Vorschriften** notwendig. Diese Transformationsregeln bilden die formalisierten Beschreibungen auf die nachgelagerte Ebene ab. MOF definiert dabei die Richtlinien für die gesamte Entwicklung. Anhand dieser Regeln wird ein UML-Metamodell definiert, aus dem die konkreten UML-Modelle abgeleitet werden (vgl. OMG MOF Specification 2003, S. 35). UML dient zur Erstellung des PIM, CWM zur Spezifikation der Metadaten für die Datenquellen. Anhand von Transformationsregeln wird das PIM in unterschiedliche PSMs umgewandelt. Nach der Spezifikation der unternehmensindividuellen Verteilungsparameter innerhalb des EDM kann der Code über entsprechende **Profile** generiert werden. Dabei ist zu beachten, dass noch nicht für alle Ebenen oder Plattformen Vorschriften für die automatische Überführung definiert sind. Der Gesamtzusammenhang der Modellierungsansichten sowie die Zuordnung zu den Modellierungsstufen und der **generativen Software-Entwicklung** wird in Abb. 2 verdeutlicht.

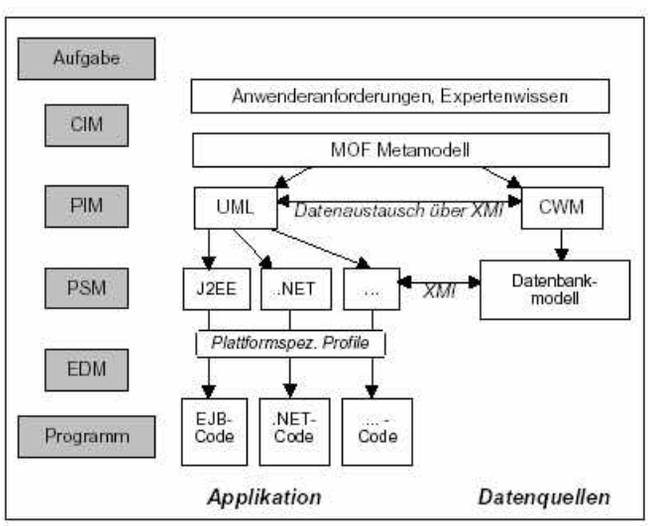


Abb. 2: Modelle und Schritte der Model Driven Architecture

Frage 1: Welche Parallelen bestehen zwischen der Programmerstellung und -ausführung mit Java und dem mehrstufigen Modell der Model Driven Architecture?

2. Komponenten der MDA

2.1. Modellierung

Zur erfolgreichen Erstellung wiederverwendbarer, plattformunabhängiger Modelle bedarf es verschiedener Vorschriften, um sowohl die eigentliche Geschäftslogik als auch die benötigten Daten- und Nachrichtenaustauschbeziehungen abbilden zu können. Für diese Aufgabenbereiche stehen im Rahmen der Model Driven Architecture die Modellierungskonzepte UML, MOF sowie CWM zur Verfügung.

Meta Object Facility (MOF)

Die Meta Object Facility beschreibt Funktionen und Beziehungen auf dem höchsten Abstraktionsniveau und definiert Strukturen für die Metamodelle weiterer Modellierungssprachen. Z.B. spezifiziert ein UML-Metamodell die Notation von UML-Modellen, die innerhalb eines Software-Projekts aufgestellt werden. Bausteine dieser abstrakten Syntax sind die für die Beschreibung der Sachzusammenhänge vorhandenen Konstrukte und deren Verknüpfungen untereinander. Deshalb wird die MOF auch als **Meta-Metamodellierungssprache** bezeichnet. Anhand der hinterlegten Vorschriften können eigene Metamodelle geschaffen oder Erweiterungen zu bestehenden Spezifikationen, beispielsweise eine unternehmensindividuelle Anpassung der UML, vorgenommen werden. In diesen Fällen ist eine automatisierte Überführung der einzelnen Stufen bis hin zur Codegenerierung nur möglich, wenn die fehlenden Transformationsregeln für die selbst erstellten Erweiterungen in den entsprechenden Code-Generatoren nachgepflegt werden (vgl. OMG MOF Specification 2003, S. 33 - 37).

Die MOF dient zudem der einheitlichen Beschreibung der **Metadaten** innerhalb eines Projekts. Bei dem Erstellen oder Erweitern von Anwendungen sowie bei der Integration von Systemen werden durch die zentrale Beschreibung der Inhalte Missverständnisse und Kommunikationsprobleme vermieden. Beispielsweise wird festgelegt, dass Datenfelder mit dem Namen „Verkaufspreis“ in allen Programmen und Datenbanken den Preis inklusive Umsatzsteuer enthalten. Bei einer fehlenden Definition und unterschiedlicher Auslegung würden übergreifende Berechnungen zu Fehlern führen. Des Weiteren werden Applikations- und Datenmodelle sowie Datentransformationsregeln über die MOF verwaltet (vgl. Frankel 2003, S. 96 - 112).

Unified Modeling Language (UML)

Die Unified Modeling Language, die bald in der Version 2.0 von der OMG verabschiedet wird, hat sich in den letzten Jahren als anerkannte Modellierungssprache insbesondere bei der objektorientierten Entwicklung etabliert. Die UML bietet dem System-Designer eine Vielzahl von Diagrammtypen mit unterschiedlichen Sichten und **Abstraktions- bzw. Detaillierungsgraden** bezüglich des zu entwickelnden Anwendungssystems. Diese Diagramme legen die Interaktion des Systems mit der Außenwelt (Anwendungsfall-, Zustandsdiagramme), die benötigten Klassen und die Beziehungen untereinander (Klassendiagramme) sowie den eigentlichen Programmablauf der Geschäftslogik (Sequenz-, Kollaborations- und Aktivitätsdiagramme) fest. Darüber hinaus stehen weitere Typen zur Definition von Objekten, Paketen, Implementierung und Verteilung sowie Komponentendiagramme zur Verfügung. Neben der Systemspezifikation dienen UML-Diagramme als Kommunikationsgrundlage zwischen Entwickler und Auftraggeber und helfen somit, die fachlichen Anforderungen exakt zu spezifizieren.

Für den Einsatz innerhalb der Model Driven Architecture muss die standardisierte UML-Notation zu UML-Profilen erweitert werden. Durch die so gewonnenen Metamodelle wird die Bedeutung der UML-Diagramme formal definiert und eindeutig. Eine automatische Transformation der Modelle in andere Modelle oder ausführbaren Code ist möglich, wenn die definierten Profile als Erweiterungen in die Werkzeuge eingepflegt werden (vgl. Frankel 2003, S. 150, 154 f.). Zur Präzisierung der UML wird die Object Constraint Language (OCL) verwendet, welche eine Beschreibung von Modellelementen in Form von Code erlaubt.

Common Warehouse Model (CWM)

Das CWM dient der Anbindung und Nutzung einheitlicher Metadaten im Bereich eines Data Warehouse. Es basiert auf dem **Message-Oriented Framework** und verbindet UML, XMI und MOF zur Spezifikation des Austausches dieser Metadaten. Ebenso wie der Einsatz der Model Driven Architecture eine Zusammenarbeit unterschiedlicher Applikationen auf verschiedenen Plattformen ermöglicht, erlaubt das CWM die Nutzung und Analyse unterschiedlicher Datenquellen durch den Aufbau eines Data Warehouse sowie Data Mining Aktivitäten. Verschiedene Datenbankschemata können aufeinander abgebildet werden.

CWM entstand aus einer Kooperation der OMG mit der Meta-Data Coalition. Analog zur Modellierung von Programmen mittels der UML erfolgt die Datenmodellierung durch das CWM unabhängig von der Implementierung. CWM erlaubt die Modellierung unterschiedlicher **Datenquellen**, beispielsweise relationale, objektorientierte, objektrelationale oder XML-basierte Datenbanken (vgl. OMG CMW Specification 2003, S. 43 - 45). Die Anbindung der Datenquellen und die Abbildung der Datenstrukturen erfolgt innerhalb des durch das CWM für den Austausch der Metadaten vorgegebenen Frameworks. Neben dem einheitlichen Zugriff auf die unterschiedlichen Datenbanken werden die Metadaten des Data Warehouse zur Analyse mittels OLAP (On-line Analytical Processing) und Data Mining sowie zur

Informations-Visualisierung verwendet. Dabei werden Datenquellen und Datenziele sowie die Transformation und Analyse von Daten unterstützt (vgl. Andresen 2000, S. 250 - 252).

2.2. Umsetzung

Auf der Implementierungsebene der Model Driven Architecture stehen unterschiedliche Middleware-Ansätze zum Aufbau verteilter Applikationen zur Verfügung. Aufgaben sind neben der Spezifikation der Kommunikation und des Datenaustausches die Bereitstellung übergreifender Dienste, z.B. Transaktionsmanagement oder Ereignisbehandlung.

XML/XMI

Die Extensible Markup Language (XML) ist eine Auszeichnungssprache, die von SGML abgeleitet wurde und die Möglichkeit bietet, eigene Auszeichnungselemente (Tags) zu definieren. Mit Hilfe von XML ist es möglich, Daten zu strukturieren und sie mit Hilfe der definierten Tags in nicht nur für Maschinen lesbare Form zu bringen. Als problematisch erweist sich dabei allerdings, dass **semantische Lücken** bestehen können, wenn Unternehmen unterschiedliche Tags für den gleichen Sachverhalt verwenden und umgekehrt. Daher bedarf es bei der Nutzung von XML einheitlicher Richtlinien, um die inhaltliche Verarbeitung der Informationen gewährleisten zu können.

Um die Daten und Datenstrukturen der Modellierungsmodelle aus MOF und CWM technisch abbilden zu können, bietet sich der Einsatz von XML an. Um das Problem der semantischen Lücke zu beheben, wurde die XML Metadata Interchange (XMI) geschaffen. Mit Hilfe dieser Spezifikation ist es möglich, Metadaten auszutauschen und in einer **einheitlichen Beschreibung** abzulegen, die allgemein gültig und verständlich ist.

CORBA

Die Common Object Request Broker Architecture (CORBA) ist eine Middleware-Plattform, welche die generelle Kopplung verteilter Anwendungen und deren Interoperabilität ermöglicht. CORBA wurde wie die Model Driven Architecture von der OMG spezifiziert. Mit dem über MOF definierten **CORBA Component Modell** (CCM) wurde ein eigenes Komponentenmodell spezifiziert, das plattform- und programmiersprachenneutral ist. CCM stellt Dienste zur Entwicklung, Implementierung, Konfiguration und Verteilung von komponentenbasierten Systemen bereit. Über die CORBA Interface Definition Language (CORBA IDL) können Anwendungen oder Komponenten, die in unterschiedlichen Programmiersprachen implementiert wurden, miteinander kommunizieren. Diese Schnittstellen-Beschreibungen werden über Generatoren in die Zielsprachen übersetzt. Die Kommunikation der Komponenten erfolgt über den Object Request Broker (ORB). Zusätzlich stellt CORBA einen Namensdienst zur Auflösung von Objektreferenzen zur Verfügung (vgl. Andresen 2003, S. 227 - 228, 264).

Java

Die Systemplattform Java von Sun Microsystems hat seit der Veröffentlichung der Java 2 Enterprise Edition eine stetig wachsende Anzahl von Anwendern verzeichnen können. Unter der Produktbezeichnung JavaOne hat Sun verschiedene Application Programming Interfaces (API) zusammengeführt, die alle erforderlichen Konzepte und Technologien zur Realisierung von Individual-Software und zur Anbindung anderer Anwendungssysteme vereinigt. Neben der Unterstützung von Datenbanken, XML oder Web Services stehen insbesondere die **Enterprise JavaBeans (EJB)** als Komponentenmodell im Mittelpunkt der Plattform. EJB ermöglichen die Entwicklung leistungsfähiger verteilter Systemarchitekturen. Die Plattform J2EE wurde dabei an den standardisierten Spezifikationen des CORBA-Modells ausgerichtet und ist damit zu CCM voll kompatibel.

Innerhalb der J2EE-Architektur werden 4 Schichten (Tier) unterschieden. Über den Client Tier werden dem Anwender Informationen ausgegeben und eine Interaktion mit dem System ermöglicht. Der Web Tier stellt die Kommunikation zwischen Client Tier und Business Tier her und übernimmt die Informationsdarstellung. Im Business Tier ist die Geschäftslogik abgelegt. Hier werden Persistenz-, Transaktions- und Zugriffsdienste verwaltet. Eine weitere Aufgabe des Business Tier ist, Komponenten des EIS-Tier (Enterprise Information Systems) anzustoßen. Diese dienen z.B. der Anbindung an Datenbanken oder ERP-Systeme (vgl. Andresen 2003, S. 257 - 258).

.NET

Microsoft bietet mit .NET eine weitere Systemplattform zur Entwicklung komplexer Anwendungen an und ist damit größter Konkurrent zu JavaOne. Dazu werden unter anderem ein eigenes Framework, Web Services sowie .NET Enterprise Server angeboten. Im .NET Framework werden Laufzeitsystem, Klassenbibliotheken und Subsysteme zur Applikationsentwicklung, beispielsweise ASP.NET, zur Verfügung gestellt. Der BizTalk Server ermöglicht die Kommunikation zwischen verschiedenen Anwendungen und Diensten über den Austausch von XML-Dokumenten und stellt verschiedene Übertragungsprotokolle zur Verfügung (vgl. Andresen 2003, S. 275 - 277).

Ein Bestandteil der .NET-Technologie, **COM+**, ist ein von Microsoft spezifiziertes Komponentenmodell. Es vereint DCOM (Distributed Component Object Model), ein programmiersprachenneutrales, aber plattformspezifisches Komponentenmodell für verteilte Anwendungen, und MTS (Microsoft Transaction Server), der das Transaktionsmanagement in verteilten Systemen übernimmt.

Frage 2: Welcher Zusammenhang besteht zwischen den Modellierungssprachen MOF, UML und CWM?

Web Services

Unter Web Services werden alle Dienste subsumiert, die auf Basis von Internetprotokollen heterogene Anwendungssysteme koppeln und über entsprechende Schnittstellen Daten austauschen. Diese Dienste können zur innerbetrieblichen Anwendungskopplung sowie zur zwischenbetrieblichen Kommunikation genutzt werden. Ebenso ist es möglich, Web Services als Informationsbausteine für Web-Seiten zur Verfügung zu stellen, über die einzelne Anwender z.B. Buchungen durchführen oder Cross-Selling-Angebote wahrnehmen können.

Web Services beruhen dabei in der Regel auf Protokollen, die mit Hilfe von XML realisiert wurden. Dazu zählen insbesondere das Simple Object Access Protocol **(SOAP)**, die Web Services Description Language **(WSDL)** und die Universal Description, Discovery and Integration **(UDDI)**. Durch SOAP wird eine Nachricht zwischen verteilten Anwendungen nach Typ und Inhalt spezifiziert. Die für die Nutzung eines Web Services notwendigen Aufruf- und Abwicklungsmodalitäten beschreibt WSDL. UDDI dient als Verzeichnisdienst zur Bereitstellung und Klassifikation der angebotenen Web Services.

2.3. Fachbereiche

OMG-Fachbereichs-Spezifikationen

Die OMG hat Arbeitsgruppen zur Entwicklung normativer UML-Modelle für einzelne Fachbereiche, beispielsweise Medizin, Finanzwesen, Transport oder Telekommunikation gebildet. Diese beschäftigen sich mit der Abbildung spezifischer Arbeitsabläufe sowie deren Operationen, Bedingungen und Schnittstellen. Für den Einsatz in einem Unternehmen innerhalb eines solchen Fachbereichs können die so modellierten Prozesse übernommen und durch individuell vorhandene Abläufe ergänzt werden.

Neben den bisher erarbeiteten Themengebieten werden von der OMG weitere Fachbereiche untersucht. Die normativen UML-Modelle sind mit technischen Design-Patterns vergleichbar. Durch die Bereitstellung von Musterprozessen kann die Entwicklung weniger zeit- und kostenintensiv erfolgen.

3. Einsatz der Model Driven Architecture

Die Model Driven Architecture kann — soweit sie bereits spezifiziert wurde und in Form von Entwicklungsumgebungen zur Verfügung steht — den kompletten Software-Lebenszyklus begleiten. Dabei ist zwischen der Modellierung und der möglichst automatischen Erzeugung von Programmcode im Sinne der generativen Software-Entwicklung zu unterscheiden.

3.1. Software-Entwicklung

Hauptanwendungsfälle für die Model Driven Architecture sind System-Design und Software-Entwicklung. Insbesondere bei umfangreichen und komplexen Anwendungssystemen wird durch die verschiedenen Abstraktionsstufen die Handhabung der einzelnen Komponenten sowie deren Interaktion und Kommunikation erheblich erleichtert. Die Trennung des fachlichen Konzepts von den technischen Spezifikationen eröffnet zudem die Möglichkeit, die Kommunikation mit dem Auftraggeber zu erleichtern. Die automatisierte Code-Generierung am Ende des Entwicklungsprozesses unterstützt die Programmierer darüber hinaus beim Übergang von den einzelnen Modellen hin zu fertigen Software-Bausteinen.

Durch **formalisierte Beschreibungen** erhöht sich zwar der Aufwand im Bereich der Anforderungsanalyse und des System-Designs, aber andererseits werden qualitativ bessere Ergebnisse in der Entwicklung und Implementierung möglich. Zudem ergeben sich aufgrund der verstärkten Wiederverwendbarkeit langfristig Zeitvorteile. Neben UML und MOF zur Modellierung kommen der Datenaustauschsprache XMI sowie den Schnittstellenbeschreibungen mit IDL besondere Bedeutung durch die Gewährleistung des Datenaustausches und der Werkzeugintegration zu.

3.2. Software-Test

Überprüfung auf Fehler

Das Konzept der Model Driven Architecture kann neben der reinen System-Entwicklung dazu genutzt werden, die erstellten Modelle und erzeugten Programm-Codes bereits frühzeitig auf Fehler und Unstimmigkeiten zu überprüfen. Durch den Einsatz von Werkzeugen, die entsprechende Modellierungs- und Transformationsvorschriften der Model Driven Architecture beinhalten, werden Entwickler bei Software-Tests unterstützt und zeit- und kostenintensive Überarbeitungen zu späteren Zeitpunkten vermieden (vgl. Hauber et al. 2003, S. 20 f.).

Die selbst entwickelten Modelle sind Instanzen der herangezogenen **Metamodelle** und können gegen diese validiert werden. So definiert das UML-Metamodell unter anderem die bei der Modellierung mit UML verwendbare Notation, beispielsweise Klassen, Attribute, Operation oder Assoziationen. Das UML-Metamodell selbst wird wiederum über die MOF spezifiziert (vgl. Völter 2003, S. 61). Somit werden im Gegensatz zur klassischen Software-Entwicklung bereits in der Phase des System-Designs Tests bezüglich der zu erstellenden Anwendungen ermöglicht. Der aus den Metamodellen generierte Code ist konform zu der in den Modellen festgelegten Architektur. Bei der Validierung kommt XMI als Austauschformat zum Einsatz.

3.3. Software-Adaption

Anpassung bestehender Systeme

Software-Adaption beinhaltet die Anpassung oder Erweiterung bestehender Anwendungssysteme. Mögliche Gründe für eine Software-Adaption mit Hilfe der Model Driven Architecture sind beispielsweise ein Wechsel der technologischen Plattform, die Erweiterung von Funktionalitäten oder eine Überprüfung des bisherigen Konzepts und die Anpassung an die veränderten Anforderungen. Dabei ist zu differenzieren, ob die bestehenden Applikationen unter Verwendung der Model Driven Architecture oder auf andere Weise erstellt wurden.

Im ersten Fall ist eine komplette Neuerstellung der (Meta-)Modelle nicht erforderlich. Es findet bei fachlichen oder konzeptionellen Modifikationen lediglich eine Anpassung in den entsprechenden Modellen statt. Änderungen bezüglich der technischen Spezifikationen erfolgen in den relevanten **Transformationsregeln**. Der anschließende Prozess der Modellüberführung und Code-Generierung verläuft analog zur Software-Erstellung.

Bei der Anpassung konventionell erstellter Software wird der mehrstufige Prozess der Model Driven Architecture entgegengesetzt durchlaufen. Die vorliegenden Programme werden in die entsprechenden Modelle rücküberführt. Von bestehenden PSM wird ein PIM abstrahiert. Dies ist kaum zu automatisieren, da die Extraktion von Geschäftsprozessen aus den bestehenden Anwendungen zu vielschichtig ist. In Abhängigkeit von der Komplexität der zu lösenden Aufgabe kann das PSM wiederum in ein CIM überführt werden. Nach Abschluss der Änderungen bzw. Ergänzungen erfolgt die schrittweise Umsetzung in ein ablauffähiges Programm analog zum Vorgehen bei der Software-Entwicklung.

3.4. Software-Integration

Data Warehouse

Unter diesem Punkt wird die Integration von Anwendungen sowie unterschiedlicher Datenquellen in ein Data Warehouse zusammengefasst. Durch die strengen formalen Vorschriften beim Erstellen der einzelnen Modelle sowie der jeweiligen Datenstrukturen können andere Datenmodelle an diesen Standards ausgerichtet werden. Von der OMG wurden bereits besondere **UML-Profile** verabschiedet, beispielsweise zur Anwendungsintegration (UML-EAI) oder zur Kopplung verteilter Komponenten (UML-EDOC) (vgl. OMG Profiles 2003). Im Bereich Data Warehouse erfolgt die Modellierung über UML und CWM, die Integration in das Warehouse über das Datenaustauschformat XMI und die Schnittstellenbeschreibung IDL. Durch die stark formalisierten Beschreibungen ist es zudem möglich, Geschäftsprozesse mit Hilfe der Model Driven Architecture zu modellieren und somit eine einheitliche Definition von Daten und Prozessen für die unterschiedlichsten Aufgaben zur Verfügung zu stellen (vgl. Allgaier/Schreier/Weise 2003, S. 28 f.).

Frage 3: Wie unterscheiden sich die Vorgehensweisen der Software-Entwicklung und Software-Adaption mit Hilfe der Model Driven Architecture?

4. Einordnung der Model Driven Architecture

Um über den Einsatz der Model Driven Architecture zu entscheiden, ist es notwendig, die grundlegenden Konzepte der OMG einordnen zu können. Diese Kenntnisse erleich-

tern dem interessierten Anwender, Potenzial und Risiken eines Software-Projekts unter Verwendung der modellgetriebenen Entwicklung abzuschätzen. Dabei ist zu beachten, dass sich die Vorteile der Model Driven Architecture aufgrund von Lerneffekten seitens der Entwickler und der Wiederverwendbarkeit von (Meta-)Modellen und Transformationsregeln unter Umständen erst in späteren Projekten einstellen.

4.1. Konzepte im Hintergrund

(Meta-)Modelle

Durch die getrennte Modellierung unterschiedlicher Abstraktionsgrade wird die Entwicklung von komplexen, verteilten Systemen mit der Model Driven Architecture erleichtert. Veränderungen können an verschiedenen Stellen vorgenommen werden, wenn die Software künftig migriert oder erweitert werden soll. Der Grad der Anpassung oder Änderung bestimmt, welche Modelle variiert werden müssen. Über entsprechende Templates oder Generator Frameworks können die Modelle teilweise automatisch ineinander überführt sowie fertiger Quelltext erstellt werden. Den Rahmen für diese Modelle bestimmen die definierten Elemente und Operationen der entsprechenden Metamodelle. Diese exakte Spezifikation ermöglicht eine Validierung der erstellten Beschreibungen.

Standards

Die Model Driven Architecture basiert auf einer Vielzahl von standardisierten Technologien sowie Beschreibungs- und Modellierungssprachen. Auf Basis dieser Spezifikationen wird die Integration von Anwendungen und Software-Komponenten erhöht sowie der Austausch von (Meta-)Daten erleichtert. Die Verwendung der verabschiedeten **Standards** bei der Programmentwicklung unterstützt zudem die langfristige Erweiterbarkeit der Systeme. Der Vorteil der generativen, modellbasierten Software-Entwicklung zeigt sich dadurch, dass bei Änderungen in der Logik oder den technischen Rahmenbedingungen nur die Modelle bzw. die Transformationsvorschriften angepasst werden müssen. Der manuelle Nachbearbeitungsaufwand innerhalb des Programms wird somit stark reduziert.

Komponenten

Durch die Spezifikation der Funktionen, Datenstrukturen und Schnittstellen wird eine **baukastenartige Erstellung** von Software unterstützt. Das relevante Expertenwissen wird in einzelne Software-Komponenten verpackt. Unter einer Komponente wird dabei die Abbildung eines spezifischen Geschäftsbereichs verstanden, die im Gegensatz zu einer Applikation alleine nicht ablauffähig ist. Zwar ist die Erstellung solcher genau spezifizierten Bausteine vergleichsweise aufwändig, durch die Wiederverwendung ergeben sich aber Kostenvorteile, da auf bereits entwickelte und getestete Komponenten zurückgegriffen werden kann. Die Vorteile liegen in der Reduktion der gesamten Entwicklungskosten und -zeit eines Software-Projekts, in einer deutlich reduzierten Fehleranfälligkeit sowie in abnehmenden Wartungskosten.

Patterns

Um den Entwicklungs- und Implementierungszyklus von Anwendungssystemen weiter zu verkürzen, bietet die Model Driven Architecture aufgrund der unterschiedlich stark abstrahierten Modelle zudem die Möglichkeit, auf bestehende Konzepte und Entwürfe zurückzugreifen. Ziel ist die Schaffung eines **Repository**, in das System-Designer fertige Ergebnisse einstellen, die wiederum von anderen Entwicklern in ihren Projekten angepasst und genutzt werden können. Diese so genannten Design-Patterns beinhalten in der Regel ausgearbeitete Algorithmen und Lösungsansätze, ohne jedoch in einer konkreten Programmiersprache umgesetzt zu sein. Durch die Verwendung von Patterns und der damit verbundenen Reduktion der Komplexität ergibt sich zusätzliches Potenzial zur Kosten- und Zeitsenkung.

4.2. Effekte der Model Driven Architecture

Die Verwendung formalisierter Modelle sowie der Rückgriff auf Transformationsregeln und Design-Patterns zur generativen Software-Entwicklung verändern grundlegend die Art und Weise, wie Software erstellt wird.

Zeit und Kosten

Die modellbasierte Entwicklung sowie die Anwendung von Design-Patterns bzw. Software-Bausteinen erlauben, wie oben bereits erläutert, eine deutliche Kosten- und Zeitreduktion. Dies ist insbesondere auf das strukturierte Vorgehen, die verbesserte Kommunikation mit dem Anwender und die verstärkte Wiederverwendung von Modellen, Algorithmen und Komponenten zurückzuführen. Demgegenüber steht ein höherer Zeitaufwand beim Einarbeiten in die Model Driven Architecture, da ein Umdenken der Software-Entwickler sowie eine Umstellung bisheriger Prozesse notwendig wird. Dieser Effekt verliert mit zunehmender Erfahrung der Mitarbeiter an Bedeutung, darf bei Einführung des modellgetriebenen Konzepts jedoch nicht vernachlässigt werden. Mit erhöhtem Zeit- und Kostenaufwand ist zudem im Rahmen der Anforderungsanalyse zu rechnen, da die Erstel-

lung der Modelle höheren Anforderungen genügen muss. Dieser Aufwand wird durch geringere Folgekosten im Rahmen der Umsetzung, der Implementierung sowie der Wartung und Pflege gerechtfertigt.

Qualität

Die Qualität der Software wird durch die **Trennung des Gesamtkonzepts in verschiedene Modellierungsschichten** erhöht, da die relevanten Spezifikationen von den für die jeweilige Ebene zuständigen Experten vorgenommen werden. Durch die strikte Formalisierung sowie die Möglichkeit, bereits im Rahmen der Anforderungsanalyse und Systemkonzeption exakte Tests durchzuführen, sinkt die Fehleranfälligkeit der künftigen Software zusätzlich. Problematisch erweisen sich die bisher fehlenden oder noch nicht abschließend standardisierten Modellierungsvorschriften und Transformationsregeln. Fehlende Spezifikationen werden derzeit durch unternehmensindividuelle oder Entwicklungswerkzeug-spezifische Ansätze ergänzt. Somit besteht die Gefahr proprietärer Systeme, wodurch die Vorteile der Standardisierung verloren gehen. Hier ist die OMG gefordert, die noch vorhandenen Lücken zu füllen und die entsprechenden Spezifikationen zu verabschieden.

4.3. Model Driven Architecture in der Praxis

Ergebnisse einer Fallstudie

Eine von The Middleware Company durchgeführte Fallstudie stellt die Verwendung von marktführenden IDEs (Integrated Development Environment) dem Einsatz von Werkzeugen der Model Driven Architecture bei der Durchführung von Software-Projekten gegenüber. Der Vergleich mit regulären, codezentrierten Programmierumgebungen ergab eine **Produktivitätssteigerung** von 35% durch die Verwendung der Model Driven Architecture bei der Entwicklung von Anwendungen. Für die Untersuchung wurden zwei gleichwertig qualifizierte Entwicklerteams beauftragt, eine J2EE-Anwendung zu programmieren. Das MDA-Team benötigte 330 Stunden zur Erstellung eines fehlerfreien Programms, das traditionelle Vorgehen erforderte 507 Stunden (vgl. Herzig 2003, S. 76 - 77). Zudem werden die **bessere Anpassbarkeit** modellgetriebener Systeme an technologische oder geschäftsbezogene Änderungen und der damit verbundene geringere Aufwand hervorgehoben. Ähnlich betont Giga, ein Tochterunternehmen von Forrester Research, im Bericht „Developing Efficiency: Moving Away from a Code-Centric Strategy for Competitive Advantage“ die Notwendigkeit, sich von bisherigen Vorgehensweisen bei der Entwicklung zu lösen. Der Verwendung von Komponenten sowie von modellbasierten Mustern wird eine höhere Effizienz bescheinigt (vgl. o.V. 2003).

Frage 4: Welches sind die wesentlichen Vorteile des Einsatzes von Metamodellen in der Anwendungsentwicklung?

5. Fazit

Mit der Model Driven Architecture wurden bekannte Konzepte der modellbasierten sowie der generativen Softwareentwicklung in ein geschlossenes Modell überführt. Durch die Betonung komponentenbasierter Systeme sowie der Verwendung von Patterns findet sie ihren Platz innerhalb der modernen Softwareentwicklung. Nicht alle bislang beworbenen Vorteile sind aber bereits umsetzbar. Die automatische Übersetzung der fachlichen Konzepte des PIM in das technologiespezifische PSM und weiter in ausführbaren Code ist noch nicht umfassend gegeben. Zum einen existieren noch keine Transformationsregeln für alle Plattformen und Modelle, zum anderen erfordern aktuelle Werkzeuge immer noch ein gewisses Maß an manueller Nachpflege und Spezifikation außerhalb des PIM.

Bereits jetzt werden aber **messbare Vorteile des modellbasierten Ansatzes** sichtbar. Die strukturierte Vorgehensweise sowie die Trennung von fachbezogenem Code und verwendeter Technologie durch die Definition von Architektur und Design erlauben eine schnellere Entwicklung. Mit der klaren Spezifikation der erstellten Software-Bausteine lassen sich Modelle, Komponenten und Teilsysteme einfacher wiederverwenden. Einheitliche Metadatenstrukturen ermöglichen eine verbesserte Integration von verteilten Anwendungen.

Literaturempfehlungen:

Allgaier, M./Schreier, U./Weise, D.: Eine Modelldrehscheibe zwischen Geschäftsprozessen und Workflows. In: OBJEKTspektrum, 2003, Heft 2, S. 33 - 35.
Andresen, A.: Komponentenorientierte Softwareentwicklung. München 2003.
Bohlen, M./Starke, G.: MDA entzaubert. In: OBJEKTspektrum, 2003, Heft 3, S. 52 - 57.

Frankel, D.: Model Driven Architecture. Applying MDA to Enterprise Computing. Indianapolis 2003.
Hauber, R./Ziegler, M./Erskine, M./Hilsenbeck, R.: Modellbasiertes Testen. In: OBJEKTspektrum, 2003, Heft 3, S. 20 - 25.
Herzig, A.: OptimalJ und MDA. In: Javamagazin, 2003, Heft 9, S. 75 - 77.
OMG: Common Warehouse Model (CWM) Specification. http://www.omg.org/cgi-bin/apps/doc?formal/03-03-02.pdf.
OMG: MDA Guide Version 1.0.1. http://www.omg.org/docs/omg/03-06-01.pdf.
OMG: Meta Object Facility (MOF) Specification. http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf.
OMG: UML Profiles. In: http://www.omg.org/mda/specs.htm#Profiles.
o.V.: Model-Driven Architecture Leads to Productivity Gain of 35%. http://www.embeddedstar.com/press/content/2003/7/embedded9707.html.
Roßbach, P./Stahl, T./Neuhaus, W.: MDA-Konzepte. Javamagazin 9/2003, S. 22 - 25.
Völter, M.: Modellbasierte Codegenerierung. In: Java Spektrum, 2003, Heft 5, S. 61 - 65.

Die Fragen werden im WISU-Repetitorium beantwortet.

Die Examensklausur aus der Wirtschaftsinformatik

Die Aufgabe wurde im Wintersemester 2001/02 von Prof. Dr. Dieter Bartmann an der Universität Regensburg als Teil der Diplomprüfung im Schwerpunkt Bankinformatik des Studiengangs Wirtschaftsinformatik gestellt. Zur Bearbeitung standen 55 Minuten zur Verfügung. Bei der Musterlösung wirkte Dipl.-Wirt.-Inform. Christian Locher mit.

Aufgabe: Im Rahmen von Basel II werden neue Vorschriften zur Eigenkapitalhinterlegung im Kreditgeschäft diskutiert. Ziel ist es, eine differenzierte Bewertung der kundenindividuellen Risikosituationen bei Kreditgeschäften zu ermöglichen.

a) Beschreiben Sie die neuen Aufgaben, vor die sich die Banken gestellt sehen. (30 Punkte)
b) Welche Veränderungen in der Kundenstruktur sind zu erwarten? Unterscheiden Sie Großbanken und kleinere Kreditinstitute. (10 Punkte)
c) Zeigen Sie, auf welche Weise hier die Bank IT-Unterstützungsfunktionen übernehmen kann. (15 Punkte)

I. Daran hätten Sie denken müssen:

Zu Teil a):

Die Zunahme der Insolvenzen, die steigende Zahl Not leidender Kredite sowie die Änderungen der aufsichtsrechtlichen Rahmenbedingungen veranlassen die Kreditinstitute, ihre Anstrengungen zur Verbesserung des Risikomanagements weiter zu forcieren. In diesem Zusammenhang sind insbesondere die neuen Vorschriften zur Eigenkapitalhinterlegung (Basel II) zu nennen.

Die bisherige Pauschalregelung (Eigenkapitalhinterlegung: 8% des Kreditvolumens) erlaubt keine Berücksichtigung der individuellen Bonität des Schuldners bei einer Kreditvergabe. Die gravierendsten Nachteile der bisherigen Regelung sind neben der Missachtung der individuellen Bonität die Vernachlässigung von konjunkturellen Schwankungen, die fehlende Berücksichtigung von Portfolioeffekten sowie die mangelnde Differenzierung des Kreditrisikos.

Basel II sieht zwei verschiedene Ansätze zur Festlegung der Mindesteigenkapitalhinterlegung vor. Im Standardansatz werden nur **externe Ratings** akzeptiert. Alle nicht gera-

Model Driven Architecture (MDA)

Die Autoren

Peter Fettke
Peter Loos

Dipl.-Wirt.-Inf. Peter Fettke,
Prof. Dr. Peter Loos,
Johannes Gutenberg-Universität Mainz,
Lehrstuhl für Wirtschaftsinformatik und BWL,
ISYM – Information Systems & Management,
55099 Mainz,
E-Mail:
{fettke|loos}@isym.bwl.uni-mainz.de,
WWW: www.isym.bwl.uni-mainz.de

1 Einführung

Die *Object Management Group* (OMG) wurde 1989 als gemeinnützige Organisation mit dem Ziel gegründet, objektorientierte Technologien zu standardisieren, um so die herstellerübergreifende Integration von Software zu erleichtern. Frühe Arbeiten der OMG führten zur *Object Management Architecture* (OMA) mit dem Kernstandard der *Common Object Request Broker Architecture* (CORBA). Die vorgelegte Spezifikation eines *Object Request Broker* beeinflusste Architektur und Entwicklung von Software, behandelte aber primär implementierungstechnische Aspekte wie Objektmodell, Typsystem und Persistenz. Eine deutliche Ausweitung der Standardisierungsgegenstände der OMG erfolgte im Jahre 1997 mit der Übernahme der *Unified Modeling Language* (UML), einer objektorientierten Modellierungssprache, die den seit Anfang der 1990er Jahre herrschenden objektorientierten Methodenpluralismus vereinheitlichte und sowohl zur Repräsentation von Entwurfs- als auch von Analyseartefakten der Systementwicklung verwendet wird. Inzwischen stehen zahlreiche weitere OMG-Standards bereit, die vielfältige Aspekte der Softwareentwicklung behandeln.

Um die Vielfalt der Standards zu beherrschen, ihre Konsistenz zu bewahren und um ein einheitliches Gesamtbild für die Softwareentwicklung bereitzustellen, treibt die OMG seit dem Jahre 2000 den Rahmenstandard der *Model Driven Architecture* (MDA) voran [OMG00; OMG01a; OMG01b; OMG03a; MSUW02]. Im Selbstverständnis der OMG werden mit der MDA die bisherigen Standardisierungsbemühungen in einem weiteren Evolutionsschritt fortgeführt. Dadurch soll ein Leitbild der Entwicklung von Software etabliert werden, das sowohl die Implementierung als auch die Analyse und den Entwurf von Systemen umfasst. Ein Charakteristikum der MDA ist die klare konzeptionelle Unterscheidung zweier Spezifikationsperspektiven: Aus einer Perspektive wird ein System *unabhängig* konkreter Technologien spezifiziert, eine andere Perspektive nutzt zur Systemspezifikation Konzepte *bestimmter* Technologien. Eine möglichst automatisierte Generierung der technologieabhängigen Spezifikation ist ein weiteres Merkmal der MDA.

2 Wesentliche Konzepte

Bild 1 zeigt die wesentlichen Konzepte der MDA [KlWB03, 15–31, 83–106]:

- *Modelle:* Modelle werden zur Repräsentation von Anwendungsdomänen oder Softwaresystemen verwendet. Man unterscheidet das *Platform Independent Model* (PIM) und das *Platform Specific Model* (PSM). Ein PIM zeichnet sich im Vergleich zum PSM dadurch aus, dass es seinen Gegenstand ohne Bezug auf eine bestimmte Technologie repräsentiert. Sowohl PIM als auch PSM sind keine absoluten Kategorien, sondern stets im Kontext einer bestimmten Technologie zu verstehen. Zu unterscheiden sind generische Plattformen (bspw. Objektorientierung, Batch-Betrieb), technolo-

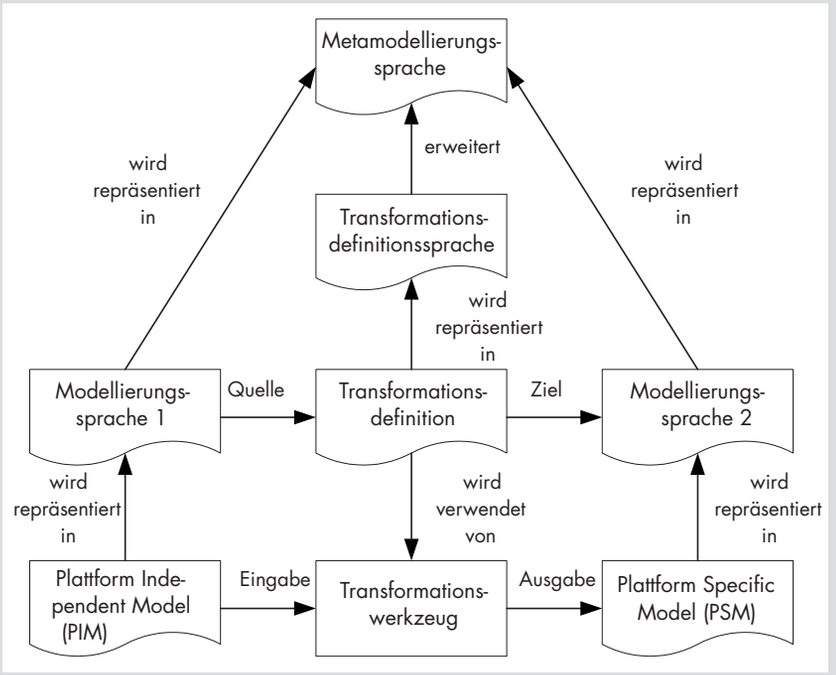


Bild 1 Wesentliche Konzepte der MDA [in Anlehnung an KlWB03, 105]

giespezifische Plattformen (bspw. CORBA, J2EE) und herstellerspezifische Plattformen (konkretes Produkt einer technologiespezifischen Plattform eines Herstellers) [OMG03a, 2–4].

- *Modellierungssprache:* Jedes Modell ist in einer Modellierungssprache repräsentiert. Von der MDA werden Modellierungssprachen bevorzugt, deren Syntax und Semantik wohldefiniert sind, sodass eine automatisierte Interpretation der Sprache möglich ist. Als Modellierungssprache kann bspw. die UML verwendet werden, wobei die MDA auch für andere Sprachen wie bspw. Petri-Netze offen ist. Die Konstrukte der Modellierungssprache und ihrer Konstruktionsregeln werden über eine *Metamodellierungssprache* definiert.
- *Transformationsdefinition:* Ein PSM soll nicht manuell erstellt, sondern möglichst automatisiert generiert werden. Derartige Transformationen werden mithilfe von *Transformationswerkzeugen* durchgeführt. Grundlage der Transformation ist die *Transformationsdefinition*, welche beschreibt, wie die Sprachkonstrukte eines PIM auf die Sprachkonstrukte des PSM abgebildet werden. Eine Transformation kann bspw. definieren, dass für jede Klasse des PIM eine Klasse im PSM generiert wird, wobei zusätzlich für jedes Attribut der Klasse Zugriffsmethoden erzeugt werden. Weitere Beispiele für Transformationen sind die Generierung von relationalen Datenbankmodellen oder Ein-/Ausgabemasken aus Klassenmodellen. Die Beschreibung der Transformationsdefinition erfolgt in einer *Transformationsdefinitionssprache*, die auf einer erweiterten Metamodellierungssprache beruht.

Einschränkend sei darauf hingewiesen, dass in Bild 1 nur ein PSM dargestellt ist, obgleich mehrere PSM aus einem PIM generiert werden können. Ebenso ist die Transformation des PSM in Programmcode nicht explizit abgebildet.

3 Wichtige Basisstandards

In der MDA werden verschiedene etablierte Standards vereint [Fran03]. Die folgende Darstellung beschränkt sich auf die Rolle der Standards für die MDA.

- *UML und Object Constraint Language (OCL):* Die UML kann als eine Modellierungssprache zur Repräsentation von PIM und PSM verwendet werden. Mithilfe der OCL können diese Modelle semantisch präzisiert werden. Ferner kann die OCL einen Beitrag zur Spezifikation von Restriktionen bei der Modelltransformation leisten.
- *Meta Object Facility (MOF):* Die MOF ist eine Sprache zur Beschreibung von Modellierungssprachen (MetaModellierungssprache), bspw. soll das Metamodell von UML 2.0 mit der MOF harmonisiert werden. Die MOF ist in sich selber definiert, sodass keine höheren Metamodellierungssprachen benötigt werden. Darüber hinaus ist Metamodellierung in der MDA relevant für die Beschreibung von Transformationsdefinitionen: Die Definition von Transformationsregeln kann sich auf die Konstrukte der Metamodelle abstützen, sodass allgemein gültige Transformationsregeln angegeben werden können.
- *XML Metadata Interchange (XMI):* XMI soll zum Austausch von Modellen zwischen verschiedenen Werkzeugen verwendet werden. XMI definiert mithilfe von XML (*Extensible Markup Language*) die Serialisierung von MOF-konformen Sprachen. Hierzu beschreibt XMI, wie die Konstrukte der MOF auf XML abgebildet werden. XMI nimmt nur Bezug auf die Meta-Modellierungssprache MOF, sodass sämtliche MOF-konforme Modellierungssprachen über XMI mit einem entsprechenden XML-Schema genutzt werden können.

4 Entwicklungsprozess mit der MDA

Der Entwicklungsprozess mit der MDA ist in Bild 2 dargestellt. Der Prozess beginnt mit einer Anforderungsermittlung, deren Ergebnis eine natürlichsprachliche Beschreibung der Systemanforderungen ist. Im Rahmen der Analyse werden die Anforderungen in ein PIM überführt. Der Systementwurf in Form eines PSM wird anschließend durchgeführt. Es folgen Implementierung und Test. Der Prozess wird durch die Auslieferung des Systems abgeschlossen.

Der Gesamtprozess gleicht einem klassischen Entwicklungsprozess. Der wesentliche Unterschied ist, dass die Transformationsschritte zwischen Analyse, Entwurf und Implementierung nicht manuell, sondern weitgehend automatisiert durch-

geführt werden. Ferner sollen Prozessiterationen stets bei der Analyse beginnen. Hierdurch kann vermieden werden, dass in frühen Phasen erstellte Systemartefakte veralten bzw. Quellcode und übrige Systemartefakte inkonsistent werden.

Insgesamt ist der Entwicklungsprozess mit der MDA nur grob umrissen und kann auf Basis verschiedener Vorgehensmodelle konkretisiert werden. Die MDA kann sinnvoll im Rahmen von Entwicklungsprozessen wie bspw. dem *Unified Software Development Process* (USDP) verwendet werden. Bei vielen der im Rahmen des USDP zu entwickelnden Artefakten handelt es sich um UML-Modelle, die teilweise ein PIM konstituieren bzw. aus diesem generiert werden könnten, indem Transformationsdefinitionen gemäß der Anforderungen des USDP definiert werden. Ferner erscheint auch der Einsatz der MDA in leichtgewichtigen Prozessen sinnvoll, da innerhalb der MDA Modelle nicht nur für Zwecke der Dokumentation erstellt werden, sondern integraler Bestandteil eines Softwaresystems sind, aus dem Teile der Implementierung generiert werden [KlWB03, 40f.].

5 Entwicklungswerkzeuge

Der Entwicklungsprozess mit der MDA ist auf leistungsfähige Softwarewerkzeuge angewiesen. Insbesondere werden folgende Werkzeugtypen benötigt [KlWB03, 38–40, 149f.]:

- *Modelleditor:* Über einen Modelleditor werden Modelle eingegeben, verändert und angezeigt. Der Modelleditor kann ebenso die Validierung der Modelle unterstützen.
- *Transformationsdefinitionseditor:* Mit diesem Werkzeug können Regeln für die Transformation von Modellen spezifiziert werden.
- *Transformationswerkzeug:* Die Durchführung von Transformationsdefinitionen übernimmt ein Transformationswerkzeug. Es ist denkbar, dass für unterschiedliche Zielplattformen bzw. Transformationen spezialisierte Werkzeuge bereitgestellt werden. Transformationswerkzeuge sind in die Unterklassen PIM-PSM-, PSM-PSM- und PSM-Quellcode-Werkzeuge zu unterscheiden.
- *Repositorium:* Im Repositorium werden neben dem Quelltext insbesondere auch Modelle sowie Transformationsdefinitionen gespeichert.

Eine Übersicht über verfügbare Werkzeuge findet sich in [OMG03b]. Es bleibt abzuwarten, wie leistungsfähig sich die Werkzeuge in der Praxis erweisen.

6 Nutzenpotenziale

Die Verwendung der MDA ist mit verschiedenen Nutzenpotenzialen verbunden [KlWB03, 9–11; Buch03]:

- *Produktivität:* Die Systementwicklung konzentriert sich künftig stärker auf die Konstruktion des PIM. Ein Produktivitätsvorteil kann sich dadurch ergeben, dass ein Teil der Implementierung automatisiert generiert wird, sodass manuelle Programmierungsaufgaben reduziert werden und Entwickler mehr Zeit für die Realisierung fachlicher Anforderungen aufbringen können.
- *Portabilität*: Durch die Definition eines plattformunabhängigen Anwendungsmodells wird es möglich, dieses auf unterschiedlichen Zieltechnologien zu implementieren, wodurch eine höhere Portabilität der Systementwicklung realisiert werden kann. Dieses Vorgehen ermöglicht es, die Vorteile neuer Technologien auszunutzen, ohne fachliche Anforderungen erneut zu implementieren.
- *Interoperabilität:* Softwaresysteme sind heute in der Regel heterogene Systeme (verschiedene Betriebssysteme, Datenbanksysteme, Middleware etc.). Die Konzepte der MDA ermöglichen es, nicht nur plattformspezifische Implementierungen zu realisieren, sondern auch Schnittstellen zwischen den verschiedenen Plattformen automatisiert zu generieren (so genannte „Bridges“). Ziel ist eine leichte Kommunikation zwischen verschiedenen Teilen eines Softwaresystems.
- *Wartung und Dokumentation:* Durch eine enge Verzahnung von frühen und späten Entwicklungsartefakten können Konsistenz- und Synchronisationsprobleme zwischen Dokumenten verschiedener Phasen vermieden werden. Die graphische Repräsentation von Systemen kann zusätzlich die Verständlichkeit der Artefakte verbessern.

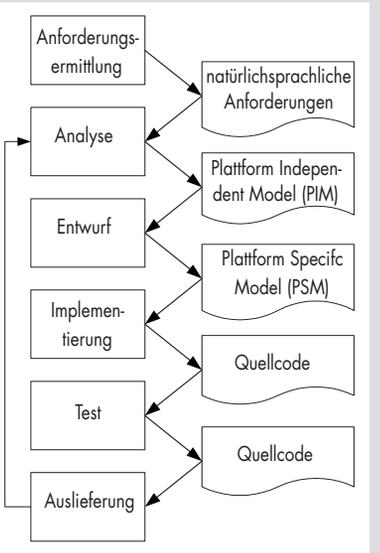


Bild 2 Entwicklungsprozess mit der MDA [KlWB03, 7]

7 Empirische Befunde

Inwieweit die Nutzenpotenziale der MDA in der Praxis der Softwareentwicklung ausgeschöpft werden können, lässt sich nur durch empirische Untersuchungen ermitteln. Bisher sind keine wissenschaftlichen Untersuchungen zu diesem Thema bekannt. Allerdings existieren mehrere Fallbeschreibungen einzelner Entwicklungsprojekte, in denen MDA-Konzepte eingesetzt wurden [OMG03c]. Die in diesen Projekten entwickelten Systeme stammen aus verschiedenen Branchen (bspw. aus dem Bankenumfeld oder dem Flugzeugbau). Die Projekte haben nicht den Charakter von Technologiestudien, sondern es handelt sich um Referenzprojekte in mittlerer Größenordnung.

In den Fallbeschreibungen wird ausgeführt, dass die hier genannten Nutzenpotenziale realisiert werden können, obgleich die zugänglichen Informationen keine Überprüfung zulassen. Beispielsweise werden in einzelnen Studien Prozentzahlen genannt, die darlegen, in welchem Umfang Quellcode automatisiert erzeugt werden konnte. Es ist allerdings zu beachten, dass eine Aussage wie „61 % des Programmcodes der EntityBeans konnten automatisiert generiert werden“ wenig über die tatsächliche Produktivitätssteigerung im Projekt aussagt, da selbst hohe Prozent-

zahlen nicht bedeuten, dass tatsächlich Implementierungsaufwände reduziert werden konnten. Zudem fällt auf, dass die zitierten Fallstudien nur positive Aspekte ansprechen, ohne auf Schwachstellen einzugehen, die bei neuen Technologien in der Regel vorhanden sind (bspw. werden in der allgemeinen Untersuchung von [MiJS02] Probleme der MDA bei der Modelltransformation aufgegriffen).

8 Bezug zur Wirtschaftsinformatik

Die Konzepte der MDA sind unabhängig von einer konkreten Anwendungsdomäne eingeführt und können bei der Entwicklung von Betriebssystemen, Embedded-Systems, Büroapplikationen u. a. Domänen angewendet werden. Für die Wirtschaftsinformatik von Relevanz sind insbesondere Anwendungen in Wirtschaft und Verwaltung. Hierbei ergeben sich viel versprechende Berührungspunkte zwischen den Konzepten der MDA und der Wirtschaftsinformatik.

Eine Kernidee der MDA, Anwendungsdomänen unabhängig von konkreten Technologien zu beschreiben, ist für die Wirtschaftsinformatik nicht neu und wurde bereits vor einiger Zeit thematisiert [LoSc95]. Beispielsweise verfolgt das von Scheer entwickelte Konzept der Architektur Integrierter Informationssysteme (ARIS) [Sche02, 38–43] ebenso die Zielsetzung, betriebswirtschaftliche Software unabhängig von vorherrschenden technologischen Entwicklungszyklen zu beschreiben. Während Scheer die Art des Übergangs von einer fachkonzeptionellen Beschreibung zur technischen Realisierung offen lässt, verfolgt die MDA den Ansatz, die technische Realisierung weitgehend aus einer fachlichen Beschreibung zu generieren.

Ein weiterer Berührungspunkt findet sich im Bereich der Referenzmodellierung. Zurzeit liegen für verschiedene Anwendungsdomänen in der Wirtschaftsinformatik Referenzmodelle vor [FeLo03], die als PIM eingesetzt werden können. Darüber hinaus können die in der Wirtschaftsinformatik bekannten Methoden zur Referenzmodellierung [FeLo02] die Ausgestaltung des Prozesses der Modelltransformation in der MDA befruchten.

Ferner kann die MDA einen Beitrag zur Integration von Informationssystemen leisten, die gemäß des Rahmenmodells von [Holt03] in verschiedene Phasen und Abstraktionsebenen gegliedert wird. Die Konzepte der MDA behandeln sowohl die Phasen Fachkonzeption, DV-Konzeption und Implementierung als auch die Abstraktionsebenen Instanzen-, Typ- und Metaebene. Folglich kann die MDA als ein umfassender Integrationsansatz verstanden werden, der alle Aspekte der Integration von Informationssystemen umfasst.

9 Aktuelle Forschungen und Entwicklungen

Abschließend werden aktuelle Forschungen und Entwicklungen im Umfeld der MDA aufgezeigt:

- *Werkzeugunterstützung:* Bisher existieren keine Softwareproduktionsumgebungen, in denen sämtliche Aufgaben der MDA unterstützt werden [KlWB03, 149f.]. Hier ergibt sich ein großes Marktpotenzial für Werkzeuganbieter.
- *Transformationsdefinition:* Sprachen zur Beschreibung von Modelltransformationen sind noch nicht standardisiert. Eine mögliche Sprache wird in [KlWB03, 93–106] vorgeschlagen. Weitere Fragestellungen ergeben sich im Hinblick auf die Spezifikation der Parametrisierbarkeit von Transformation, um in verschiedenen Kontexten (bspw. maximale Geschwindigkeit vs. minimaler Speicherverbrauch) unterschiedliche PSM generieren zu können.
- *Standardisierte Architekturen:* Neben den zuvor angesprochenen methodischen Aspekten der Transformation ergeben sich Fragen zu ihrer inhaltlichen Ausgestaltung. Es ist zu erwarten, dass sich in vielen Bereichen einheitliche Transformationsmuster herauskristallisieren, sodass Basisarchitekturen für bestimmte Systemklassen bereitstehen, die projektspezifisch zu erweitern sind. In diese Richtung entwickelt sich zurzeit bspw. der Ansatz der *Convergent Architecture* [Hube02].
- *Ausführbare Modelle:* Durch die Anreicherung der Semantik der Modellierungssprachen wird es möglich, die Ausdruckmächtigkeit der Modellierungssprache zu erhöhen, um so den Entwicklungsprozess weiter zu automatisieren und zu beschleunigen. Erste Ansätze liegen im Bereich der *Executable UML* [MeBa02] vor.
- *Standardisierte Domänenmodelle*: Seit einigen Jahren werden verschiedene Domänenmodelle innerhalb der *OMG Domain Task Forces* entwickelt, die zurzeit an die Erfordernisse der MDA angepasst und erweitert werden. Hierdurch können PIM für bestimmte Domänen standardisiert werden.

Insgesamt stellt die MDA einen technologiegetriebenen Ansatz einer modellorientierten Softwareentwicklung dar, der den in der Wirtschaftsinformatik primär anwendungsgetriebenen Ansatz einer modellbasierten Beschreibung von Informationssystemen ergänzt. Gleichzeitig kann insbesondere die wirtschaftsinformatische Forschung der Referenzmodellierung sowohl inhaltlich mithilfe von Referenzmodellen als auch methodisch – bspw. mit Methoden zur konfigurativen Referenzmodellierung [BDKK02] – interessante Impulse für die Weiterentwicklung der MDA geben.

Literatur

[BDKK02] *Becker, Jörg; Delfmann, Patrick; Knackstedt, Ralf; Kuropka, Dominik:* Konfigurative Referenzmodellierung. In: *Becker, Jörg; Knackstedt, Ralf* (Hrsg.): Wissensmanagement mit Referenzmodellen. Konzepte für die Anwendungssystem- und Organisationsgestaltung. Berlin et al. 2002, S. 25–144.

[Buch03] *Bucholdt, Christian:* Ökonomische Entscheidungskriterien für den Einsatz der MDA. In: OBJEKTspektrum (2003) 2, S. 20–25.

[FeLo02] *Fettke, Peter; Loos, Peter:* Methoden zur Wiederverwendung von Referenzmodellen – Übersicht und Taxonomie. In: *Becker, Jörg; Knackstedt, Ralf* (Hrsg.): Referenzmodellierung 2002 – Methoden – Modelle – Erfahrungen. Tagungsband zur 6. Fachtagung Referenzmodellierung 2002 im Rahmen der MKWI 2002 in Nürnberg (zugl. Arbeitsbericht des Instituts für Wirtschaftsinformatik der Westfälischen Wilhelms-Universität Münster Nr. 90, ISSN 1438–3985). Münster 2002, S. 9–33.

[FeLo03] *Fettke, Peter; Loos, Peter:* Classification of reference models – a methodology and its application. In: Information Systems and e-Business Management 1 (2003) 1, S. 35–53.

[Fran03] *Frankel, David S.:* Model Driven Architecture – Applying MDA to Enterprise Computing. Indianapolis, Indiana, USA 2003.

[Holt03] *Holten, Roland:* Integration von Informationssystemen. In: Wirtschaftsinformatik 45 (2003) 1, S. 41–52.

[Hube02] *Hubert, Richard:* Convergent Architecture: Building Model-Driven J2EE Systems with UML. Chichester et al. 2002.

[KlWB03] *Kleppe, Anneke; Warmer, Jos; Bast, Wim:* MDA Explained: The Model Driven Architecture: Practice and Promise. Boston et al. 2003.

[LoSc95] *Loos, Peter; Scheer, August-Wilhelm:* Vom Informationsmodell zum Anwendungssystem – Nutzenpotentiale für den effizienten Einsatz von Informationssystemen. In: *König, Wolfgang (Hrsg.):* Wirtschaftsinformatik '95 – Wettbewerbsfähigkeit, Innovation, Wirtschaftlichkeit. Heidelberg 1995, S. 185–201.

[MeBa02] *Mellor, Stephen J.; Balcer, Marc J.:* Executable UML: A Foundation for Model-Driven Architecture. Boston et al. 2002.

[MiJS02] *Miguel de, Miguel; Jourdan, Jean; Salicki, Serge:* Practical Experiences in the Application of MDA. In: *Jézéquel, Jean-Marc; Hussmann, Heinrich; Cook, Stephen (Hrsg.):* „UML" 2002 – The Unified Modeling Language, Model Engineering, Concepts, and Tools, 5th International Conference, Dresden, Germany, September 30–October 4, 2002, Proceedings. Berlin et al. 2002, S. 128–139.

[MSUW02] *Mellor, Stephen J.; Scott, Kendall; Uhl, Axel; Weise, Dirk:* Model-Driven Architecture. In: *Bruel, Jean-Michel; Bellahsene, Zohra (Hrsg.):* Advances in Object-Oriented Information Systems, OOIS 2002 Workshops, Montpellier, France, September 2, 2002, Proceedings. Berlin et al. 2002, S. 290–297.

[OMG00] *OMG:* Model Driven Architecture, White Paper, Draft 3.2, November 27, 2000. o. O. 2000. ftp://ftp.omg.org/pub/docs/omg/00–11-05.pdf, Abruf am 2003-06-13.

[OMG01a] *OMG:* Model Driven Architecture (MDA), document number ormsc/2001-07-01. o. O. 2001. http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01, Abruf am 2003-06-13.

[OMG01b] *OMG:* Developing in OMG's Model-Driven Architecture – Object Management Group White Paper, November 2001, Revision 2.6, document number omg/2000-11-05. o. O. 2001. ftp://ftp.omg.org/pub/docs/omg/01-12-01.pdf, Abruf am 2001-06-13.

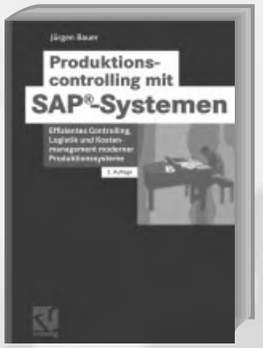
[OMG03a] *OMG:* MDA Guide Version 1.0, omg/2003-05-01. o. O. 2003. http://www.omg.org/cgi-bin/doc?mda-guide, Abruf am 2003-06-13.

[OMG03b] *OMG:* Committed Companies and Their Products. http://www.omg.org/mda/committed-products.htm, Abruf am 2003-06-13.

[OMG03c] *OMG:* Success Stories. http://www.omg.org/mda/products_success.htm, Abruf am 2003-06-13.

[Sche02] *Scheer, August-Wilhelm:* ARIS: Vom Geschäftsprozeß zum Anwendungssystem. 4. Aufl., Berlin et al. 2002.

State-of-the-Art für rationelles, schlankes Produktionscontrolling



Der Inhalt

State-of-the-Art für ein rationelles, schlankes Produktionscontrolling unter Einsatz der Module PP und CO von SAP R/3. Die erste Auflage war bereits sehr erfolgreich, die neue Auflage wurde aktualisiert und erweitert. Das Kapitel "Wirtschaftliche Lagerhaltung" wurde verstärkt, das Thema "Value Production" (zur langfristigen Wertsteigerung und Cash-Orientierung) ist hinzugekommen.

Jürgen Bauer

Produktionscontrolling mit SAP®-Systemen

Effizientes Controlling, Logistik- und Kostenmanagement moderner Produktionssysteme

2. akt. u. erw. Aufl. 2003. XIV, 264 S. mit 122 Abb. (IT-Professional, hrsg. von Dohmann, Helmut/Fuchs, Gerhard/Khakzar, Karim) Br. € 39,90

ISBN 3-528-15773-9

FAX-Bestellung 0611-7878-439

Ja, hiermit bestelle ich:

☐ **Jürgen Bauer**
Produktionscontrolling mit SAP®-Systemen
2. Aufl. 2003. € 39,90 (zzgl. Versand € 3,26)
ISBN 3-528-15773-9

Vorname/Name **321 03 005**

Firma

Straße

PLZ/Ort

Telefon/Fax

Unterschrift Datum

vieweg

Abraham-Lincoln-Str. 46
65189 Wiesbaden
www.vieweg.de
Fax: 0611.7878-439

Änderungen vorbehalten

Oft werden auch Fallstudien aus vorangegangenen Semestern übernommen und nur leicht modifiziert in aktuellen Veranstaltungen verwendet. Dafür liegt der größte Aufwand in der **Aktualisierung der Zeitangaben**. Der Generator besitzt daher die Möglichkeit, alle Zeitangaben innerhalb einer Fallstudie um einen beliebigen Zeitraum zu verschieben.

Frage 4: Warum erweist sich die Entwicklung der Fallstudien als interaktiver Prozess, bei dem mehrere Durchläufe notwendig sind?

3. Ausblick

Der Fallstudiengenerator wurde bereits bei der Entwicklung der Fallstudien für die Veranstaltungen im Sommersemester 2002 verwendet. Auch wenn es noch zu kleineren Problemen kam, die oftmals darauf zurückzuführen waren, dass der Umgang mit dem neuen Tool erst noch erlernt werden musste, war doch bereits eine erhebliche Zeitersparnis feststellbar. So war es jetzt innerhalb weniger Stunden möglich, die gesamte Fallstudie umzustellen, während dies bisher aufgrund des hohen Arbeitsaufwandes nur in Ausnahmefällen durchgeführt wurde. Der Leistungsumfang des Generators orientiert sich bisher an den Anforderungen der eingangs erwähnten Veranstaltungen. Allerdings ist das gesamte Design so ausgelegt, dass mit geringem Aufwand weitere Basisparameter und insbesondere Szenarien hinzugefügt werden können. Auch ist es ein Leichtes, die Unterstützung weiterer relationaler Datenbanken in das Programm zu integrieren. So werden bei der Generierung weiterer Fallstudien sicherlich noch Erkenntnisse gesammelt, die in die zukünftige Entwicklung des Programms einfließen können.

Literaturempfehlungen:

Erskine, J.A./Lenders, M.R./Mauffette-Leenders, L.A.: Teaching with Cases. London 1981.
Ferstl, O./Sinz, E.: Grundlagen der Wirtschaftsinformatik. 4. Aufl., München et al. 2001.
Kloock, J./Sieben, G./Schildbach, T.: Kosten- und Leistungsrechnung. 8. Aufl., Düsseldorf 1999.
Schierenbeck, H. : Grundzüge der Betriebswirtschaftslehre. 13. Aufl., München et al. 1998.
Stahlknecht, P./Hasenkamp, U.: Einführung in die Wirtschaftsinformatik. 10. Aufl., Berlin et al. 2002.
Steinhausen, D.: Simulationstechniken. München 1994.

Die Beantwortung der Fragen erfolgt im WISU-Repetitorium.

Hauptstudium

Werkzeuge zur Modellierung von Geschäftsprozessen

Dr. Markus Nüttgens, Saarbrücken/Trier

Da die Modellierung von Geschäftsprozessen immer wichtiger wird, haben auch die entsprechenden Werkzeuge zugenommen. Es gibt jedoch nur wenige Untersuchungen, die sich mit ihrer Leistung befassen. Eine umfassenden Kategorisierung soll bei der Auswahl helfen.

1. Rahmenkonzept zur Kategorisierung für GPM-Modellierungswerkzeuge

Markt für GPM-Modellierungswerkzeuge

Das Angebot an Modellierungswerkzeugen zum Geschäftsprozessmanagement (GPM) hat sich seit Beginn der 1990er Jahre zu einem eigenständigen Marktsegment entwickelt. Eine jährlich veröffentlichte Studie von Gartner Research schätzt das globale **Marktvolumen gegenwärtig auf über 500 Mio. US-$** und prognostiziert ein durchschnittliches **Marktwachstum von ca. 20%** für die kommenden Jahre. Eine weitere Prognose betrifft die Anzahl der kommerziell verfügbaren Produkte. Demnach soll sich diese

Anzahl von derzeit 35 Produkten in den kommenden Jahren tendenziell halbieren (vgl. Gartner Research 2001; Gartner Research 2002).

Untersuchungsstruktur

Eine mögliche Untersuchungsstruktur umfasst die **Werkzeugkategorien Visualisierung**, **Modellierung**, **Simulation**, **Workflow-Management** und **CASE** (vgl. Bullinger/ Schreiner 2001). In der Praxis lässt sich diese Kategorisierung allerdings kaum trennscharf anwenden. Insbesondere die Begrifflichkeit des Modellierungswerkzeugs ist oftmals — zumindest implizit — in allen fünf Kategorien enthalten.

Es wird daher hier der Begriff **Modellierungswerkzeug im weiteren Sinne** verwendet. Er umfasst die Aspekte Visualisierung, Modellierung und Simulation als integriertes Leistungsmerkmal eines Modellierungswerkzeugs. Die Anforderungen an Workflow-Management-Systeme (vgl. Wettstein 1999) und an CASE-Systeme (vgl. Herzwurm/Müller 1997) werden als Schnittstellenkonzepte zur Kopplung mit Fremdsystemen eingeführt und daher nicht weiter vertieft.

Evaluierungskonzept

Nachfolgend wird ein Rahmenkonzept entworfen, welches allgemeingültige Kriterien in eine spezifische Struktur für Modellierungswerkzeuge zum Geschäftsprozessmanagement integriert. Dies betrifft auch die **allgemeinen Kriterien für Qualitätsanforderungen an Softwaresysteme** nach DIN 66272 (ISO/IEC 9126) (DIN 1994) und **speziell für Anwendungssoftware** nach DIN ISO/IEC 12119 (früher: DIN 66 285) (DIN 95). Somit wird eine für diesen Anwendungszweck aussagekräftigere und redundanzärmere Darstellung erreicht. Nur schwer messbare Kriterien wie z.B. die Erlernbarkeit, Verständlichkeit oder Bedienfreundlichkeit werden hier nicht weiter thematisiert, bieten aber interessante Ansatzpunkte für weitere vertiefende Forschungsarbeiten.

Frage 1: Welcher Nutzen liegt in einer Kategorisierung von GPM-Modellierungswerkzeugen?

Hauptkategorien

Das Rahmenkonzept wurde in einem gegenläufigen Bottom-up- und Top-down-Verfahren abgeleitet und gliedert sich in die fünf Hauptkategorien „Produkt und Preismodell“, „Hersteller und Kundenbasis“, „Technologie und Schnittstellen“, „Methodik und Modellierung“ und „Anwendungen und Integration“ (vgl. Abb. 1).

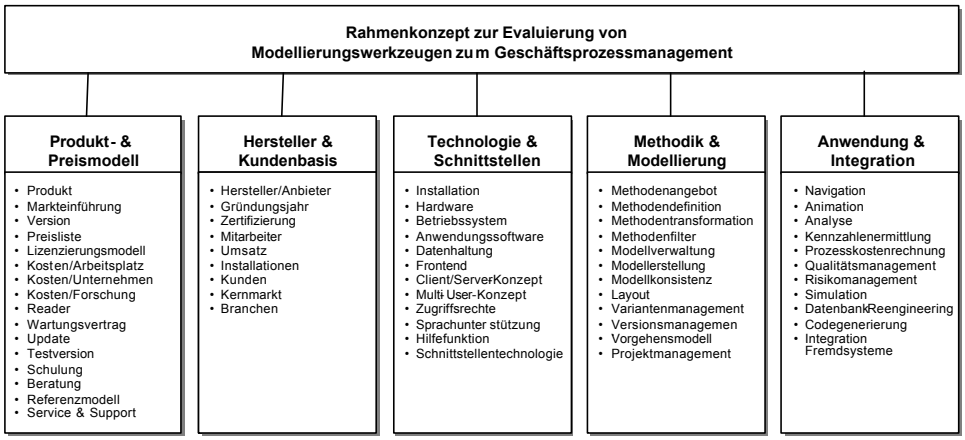


Abb. 1: Kategorisierung von GPM-Modellierungswerkzeugen

Unterkategorien

Die Hauptkategorien sind über mehrstufige Unterkategorien weiter operationalisiert und umfassen auf der Detailebene insgesamt ca. 350 Einzelmerkmale. Diese werden nachfolgend auf einer mittleren Aggregationsebene erläutert. Während die ersten drei Hauptkategorien eher allgemeine und anwendungsunabhängige Aspekte thematisieren, sind die verbleibenden Hauptkategorien auf spezifische Merkmale von Modellierungswerkzeugen ausgerichtet.

2. Produkt- und Preismodell

Im Produkt- und Preismodell werden die wesentlichen leistungs- und kostenbezogenen Eckdaten eines Modellierungswerkzeugs beschrieben. Mit der Zunahme des Funktionsumfangs bieten die Hersteller vermehrt Produktvarianten an. Die Produkt- und Preisdifferenzierung erfolgt dann über die Bereitstellung einer Basisversion, welche entweder um eine Paketlösung (Suite) oder weitere Einzelkomponenten erweitert werden kann. Die wesentlichen produktbezogenen Beschreibungsmerkmale sind:

Angebotsbezogene Unterkategorien

- **Produkt:** Da sich nur wenige Anbieter an weitgehend selbsterklärende Namenskonventionen halten, sind die Benennungen der jeweiligen Produktvarianten, der Leistungsumfang und das zugehörige Preismodell oft stark erklärungsbedürftig und erschweren eine Evaluation.
- **Markteinführung:** Das Datum der Markteinführung ist ein wichtiger Hinweis für den Reifegrad eines Produktes. Die meisten Modellierungswerkzeuge wurden zu Beginn der 1990er Jahre im Markt eingeführt und weisen heute mit ca. 5-10 Jahren Entwicklungszeit einen hohen Reifegrad auf.
- **Version:** Die aktuelle Versionsnummer und der Zeitpunkt des letzten Updates ist insofern von Bedeutung, als üblicherweise wesentliche technologische Neuerungen mit der Vergabe voller Versionsnummern verbunden sind. Eine versionsbedingte Konvertierung von Datenstrukturen kann mit einem hohen (Nach-)Bearbeitungsaufwand verbunden sein.
- **Preisliste:** Die Preislisten der Hersteller sind im Regelfall nicht öffentlich zugänglich und werden erst nach einer Kontaktaufnahme mit der Vertriebsorganisation weitergereicht. Aus Sicht potenzieller Kunden besteht bei Kenntnis der allgemeinen Marktsituation ein bedeutender preispolitischer Verhandlungsspielraum.
- **Lizenzierungsmodell:** Beim Lizenzierungsmodell ist es wesentlich, ob die Anzahl der Lizenzen personenbezogen (named user) oder mengenbezogen (concurrent user) vergeben wird.

Kostenbezogene Unterkategorien

- **Kosten/Arbeitsplatz:** Zur Abschätzung des Preismodells können die Kosten für den minimalen und den maximalen Funktionsumfang pro Arbeitsplatz herangezogen werden. Hierbei kann sich der Preis auf den Kauf, die Miete oder eine Application-Service-Providing-Lösung (ASP) beziehen. Der Preis für die Miete wird meist prozentual zum Kaufpreis angesetzt und wird üblicherweise im Falle eines späteren Kaufes angerechnet. Mit der fortschreitenden Entwicklung internetbasierter Modellierungswerkzeuge werden zunehmend auch ASP-Lösungen als alternatives Technologie- und Geschäftsmodell angeboten. Das Preisspektrum für eine Arbeitsplatzlizenz reicht je nach Modellierungswerkzeug und Funktionsumgang von 250 bis 45.000 EUR.
- **Kosten/Unternehmen:** Unternehmenslizenzen sind meist eine attraktive Alternative zu personen- oder mengenbezogenen Lizenzmodellen. Hier zeigt sich in der Praxis, dass durchweg keine offiziell festgelegten Preismodelle für Unternehmenslizenzen existieren und ein enormes Verhandlungspotenzial besteht. Unternehmenslizenzen werden oftmals auch unter marktpolitischen Gesichtspunkten ausgehandelt und sind Gegenstand strategischer Unternehmenspartnerschaften.
- **Kosten/Forschung und Lehre:** Aus Sicht von Forschungs- und Lehreinrichtungen spielt es eine wichtige Rolle, mit welchem Preisnachlass Lizenzen für Arbeitsplätze in der Forschung und Lehre bereitgestellt werden. Hierbei zeichnet sich der Trend zu ASP-Lösung ab, was insbesondere auch den technischen Betreuungsaufwand stark vereinfacht.
- **Reader:** Die Verfügbarkeit eines kostenfreien Reader/Viewer ist ein wichtiges Leistungsmerkmal. Mit einer Navigationskomponente im Stile eines Reader/Viewer können Modellinhalte ohne den vollen Funktionsumfang der Modellierungskomponente gelesen werden und sind damit einer beliebigen Zielorganisation zugänglich. Obwohl technisch mit geringem Aufwand zu realisieren, ist ein derartiges Feature preispolitisch nicht im Interesse der Anbieter. Es werden aber zunehmend mehr oder weniger kostenpflichtige Zusatzprodukte zur Erzeugung von webbasierten Exportmodellen angeboten, welche dann mit einem Webbrowser gelesen werden können.
- **Wartungsvertrag:** Wartungsverträge beinhalten wesentliche Kostenbestandteile von Modellierungswerkzeugen. Oftmals sind neben dem Versions-Update auch spezielle Services enthalten. Bei gängigen Modellierungswerkzeugen belaufen sich die Wartungskosten in Abhängigkeit zum Produkt- und Leistungsumfang zwischen 8-20% des Listenpreises/Jahr.
- **Update:** Liegt kein Wartungsvertrag vor, so muss für eine neue Version des Modellierungswerkzeugs im Regelfall ein Preis für ein Update entrichtet werden. Die Kosten hierfür belaufen sich auf ca. 50% des vollen Listenpreises.
- **Testversion:** Demoversionen dienen Testzwecken und sind im Regelfall kostenfrei im Internet oder gegen eine geringe Kostenpauschale auf CD erhältlich. Diese Versionen haben aber meist einen sehr eingeschränkten Funktionsumfang und sind zeitlich befristet. Somit ist die Aussagekraft für Evaluationen und Tests stark eingeschränkt.

Dienstleistungsbezogene Unterkategorien

- **Schulung:** Bei Trainings- und Schulungsangeboten ist relevant, inwiefern die Angebote öffentlich oder als Inhouse-Veranstaltungen angeboten werden und welchen Umfang Standardangebote (Basiskurse, Vertiefungskurse, Spezialkurse) haben. Der Preis für

Schulungen kann analog zum minimalen und maximalen Funktionsumfangs ermittelt werden. Dementsprechend können Preislinien für Schulungen zur Vermittlung von Basiswissen (entsprechend einer Basisversion) oder von Expertenwissen (entsprechend einer Version mit vollem Funktionsumfang) ermittelt und verglichen werden.

- **Beratung:** Der Preis für Beratungsdienstleistungen im Umfeld des Einsatzes eines Modellierungswerkzeugs ist ebenfalls ein wesentlicher Kostenfaktor. Hierbei ist von Interesse, ob auch pauschalierte Standardberatungen als Paketlösung angeboten werden (Einführungsberatung, Basisberatung usw.).
- **Referenzmodell:** Referenzmodelle sind Modelle, welche von Werkzeuganbietern oder Partnerunternehmen in der Form von Templates angeboten werden. Referenzmodelle sind entweder beim Kauf im Leistungsumfang eines Modellierungswerkzeugs als Bibliothek enthalten oder müssen zusätzlich kostenpflichtig erworben werden. Referenzmodelle sind entweder branchenbezogen (Industrie, Handel, Banken usw.) oder orientieren sich an bereits implementierten Standardsoftwareprodukten (SAP R/3, Oracle Applications usw.).
- **Service und Support:** Im Bedarfsfall ist es wichtig, dass ein Werkzeuganbieter über ein leistungsfähiges Customer Care Center verfügt (Hotline, „Call me back“, E-Mail, FAQs, Download-Area, Newsgroup, Bulletin-Board, Mailing-Liste, Chat usw.).

Frage 2: Welche Beschreibungsmerkmale können zur Charakterisierung der Hauptkategorie „Produkt- und Preismodell“ von GPM-Modellierungswerkzeugen herangezogen werden?

3. Hersteller und Kundenbasis

Die Hauptkategorie zur Beschreibung der hersteller- und kundenbezogenen Daten umfasst folgende Beschreibungsmerkmale:

Angbieterbezogene Unterkategorien

- **Hersteller/Anbieter:** Da es sich um einen junges und dynamisches Marktsegment handelt, ist es durchaus üblich, dass sich Herstellernamen und Adressen aufgrund von Aufkäufen, Zusammenschlüssen oder Produktübernamen im Zeitverlauf ändern. Die Vertriebsstruktur kann im Direktvertrieb, über Vertriebspartner oder als Mischform organisiert sein.
- **Gründungsjahr:** Auf der Grundlage des Alters eines Unternehmens können bedingt Rückschlüsse auf die Professionalität und Marktpräsenz gezogen werden.
- **Zertifizierung:** Die Zertifizierung der Organisationsabläufe eines Werkzeugherstellers nach ISO 9000 ff. gibt ebenfalls indirekt Aufschlüsse über die Qualität des Entwicklungsprozesses.
- **Mitarbeiter:** Die Anzahl der Mitarbeiter (weltweit) reflektiert ebenfalls implizit die Nachhaltigkeit und Professionalität der Werkzeugentwicklung. Hierbei ist zwischen Entwicklern (insbes. Programmierer) und sonstigen Mitarbeitern (Vertrieb, Beratung, Verwaltung usw.) zu unterscheiden. Die Zahl der Entwickler schwankt über alle Werkzeuganbieter hinweg zwischen unter 10 und über 100 Mitarbeitern.

Marktbezogene Unterkategorien

- **Umsatz:** Der Umsatz/Geschäftsjahr, aufgeschlüsselt nach dem Gesamtumsatz und dem Anteil an Lizenzen und Wartung, charakterisiert ebenfalls die Durchsetzungsfähigkeit eines Werkzeuganbieters am Markt.
- **Installationen:** Die Anzahl an Installationen ist ein Indiz für den Verbreitungsgrad und die Marktakzeptanz eines Modellierungswerkzeugs. Hierbei kann auch der relative Marktanteil zwischen den Produkten als Kenngröße ermittelt werden.
- **Kunden:** Die Anzahl der Kunden korreliert im Regelfall mit der Anzahl der Installationen. Ist die Relation Installation/Kunden groß, deutet dies auf eine breite Kundenbasis aus dem Segment der Großunternehmen und Konzerne hin. Ausgewählte Referenzkunden sind ebenfalls ein wichtiger Indikator für das Einsatzspektrum und die Akzeptanz eines Werkzeugs. Die Angabe von Referenzkunden ermöglicht es potenziellen Kunden, diese unabhängig von Vertriebsorganisationen zu kontaktieren und konkrete Einsatzerfahrungen zu hinterfragen.
- **Kernmarkt:** Ein weiterer Aspekt ist der Kernmarkt eines Anbieters. Dieser wird als prozentualer Anteil eines Marktes am jeweiligen Gesamtumsatz/Geschäftsjahr (z.B. Europa 60% [davon Deutschland 50%], USA 25%, Rest 15%) ermittelt. Der Kernmarkt gibt einen wichtigen Hinweis auf verfügbare Dienstleistungen und Ressourcen im Bedarfsfall (Service, Support, Beratung, Expertise usw.).
- **Branchen:** Der Einsatzhäufigkeit bezüglich bestimmter Branchen ist analog zum Merkmal Kernmarkt ein wichtiges Indiz für das Einsatzspektrum und die fachliche Ausrichtung eines Modellierungswerkzeugs.

Frage 3: Welche Beschreibungsmerkmale können zur Charakterisierung der Hauptkategorie „Hersteller und Kundenbasis“ von GPM-Modellierungswerkzeugen herangezogen werden?

4. Technologie und Schnittstellen

Die einem Modellierungswerkzeug zugrunde liegende Technologie und Schnittstellenkonzeption beeinflusst maßgeblich die Leistungsfähigkeit und das Anwendungsportfolio von Modellierungswerkzeugen. Die wesentlichen Beschreibungsmerkmale sind:

Plattformbezogene Unterkategorien

- **Installation:** Neben der Beurteilung der Qualität des technischen Installationsvorgangs (Volumen, Zeitdauer, Problemlösungsmechanismen usw.) können verschiedene Verfahren zum Kopierschutz und zur Lizenzvergabe unterschieden werden. Diese Verfahren basieren entweder auf einem Hardware-Dongle, einem generierten Key oder einer Seriennummer. Ein weitere Möglichkeit besteht darin, einem Lizenzmodell nach dem Open-Source-Konzept zu folgen.
- **Hardware:** Die Anforderungen an die Hardware werden üblicherweise als Mindestanforderung formuliert und betreffen die Prozessorleistung (Taktrate), den Hauptspeicher (RAM) und das Speichervolumen auf der Festplatte.
- **Betriebssystem:** Im Regelfall sind marktgängige Modellierungswerkzeuge auf allen windowsbasierten Betriebssystemplattformen (WIN 95/98/ME/NT/2000/XP) lauffähig. Eine weitere Alternative ist das Betriebssystem Linux, welches aber bislang auf dem Markt für Modellierungswerkzeuge praktisch keine Rolle spielt.
- **Anwendungssoftware:** Neben der Betriebssystem-Software ist zu prüfen, inwiefern Softwarekomponenten, die nicht bereits im Leistungsumfang eines Modellierungswerkzeugs enthalten sind, für dessen Betrieb vorausgesetzt werden (Bürokommunikation, Datenbank, Webbrowser, Programmierumgebung usw.).
- **Datenhaltung:** Im Rahmen der Datenhaltung lassen sich wesentliche technologische Leistungsunterschiede erkennen. Die Datenhaltung kann in der Form eines Filesystems oder eines Datenbanksystems (Oracle, Sybase, Poet usw.) realisiert sein. Eng mit dem Datenhaltungskonzept sind auch Funktionen zum Datenmerge, zur Datenkonsolidierung, zur Autospeicherung, zu Backup-Konzepten und zur Rücknahmefunktion realisiert. Beim Datenmerge werden Modellinhalte verschiedener Datenhaltungssysteme integriert. Bei der Datenkonsolidierung wird das Ziel verfolgt, homonyme und/oder synonyme Datenobjekte zu identifizieren und abzugleichen. Autosicherungs- und Backup-Konzepte gewährleisten eine konsistente Datensicherung und eine Rücknahmefunktion revidiert Arbeitsschritte bis hin zu einem festen Wiedereinsatzpunkt.

Benutzerbezogene Unterkategorien

- **Frontend:** Das Frontend marktgängiger Modellierungswerkzeuge basiert auf windows- und/oder webbasierten Technologien. Hieraus leiten sich auch die jeweiligen ergonomischen Anforderungen ab. Funktionen zur freien Initialisierung, Positionierung und Konfiguration der Fenster und Symbolleisten ermöglichen eine benutzerspezifische Adaption des Frontends.
- **Client/Server-Konzept:** Zusätzlich zur Einplatz-Version ist die Verfügbarkeit einer Server-Version ein weiteres wichtiges Leistungsmerkmal. Hiermit kann eine technologische Entkopplung von Benutzerschnittstelle, Modellierungssoftware und Datenhaltung erreicht und ein zweistufiges (two tier) oder dreistufiges (three tier) Client/Server-Konzept realisiert werden. Dies ermöglicht auch die Skalierung großer vernetzter Installationen.
- **Multi-User-Konzept:** Die Unterstützung von Konzepten für einen Mehrbenutzerbetrieb sind bei verteilten Modellierungsprojekten von besonderer Bedeutung.
- **Zugriffsrechte:** Ein weiteres wesentliches Feature ist die Verwaltung von Zugriffsrechten auf der Benutzer- und/oder Benutzergruppenebene. Hier kann der Daten-, Funktions- und Methodenzugriff unterschieden werden.
- **Sprachunterstützung:** Insbesondere in international agierenden Unternehmen ist eine leistungsfähige Sprachunterstützung von entscheidender Bedeutung. Dies betrifft die Verwaltung von die Oberflächen-, Methoden- und Modellsprachen. Features wie Rechtschreibprüfung und Thesaurus ergänzen diesen Funktionsumfang.
- **Hilfefunktion:** Umfassende Hilfefunktionen sind bei den Modellierungswerkzeugen entscheidend für die Akzeptanz. Hierzu zählen standardmäßig die Erst-Benutzer-Hilfe, Direkthilfe, Hilfethemen, Hilfe-Suchfunktion und Assistenten. Aber auch Methodenhandbuch, Entwicklerhandbuch, aussagekräftige Beispiele (Demos) und ein Homepageverweis vereinfachen den Einstieg und Umgang mit einem Modellierungswerkzeug.

— **Schnittstellentechnologie:** Mit dem zunehmenden Leistungsumfang und der fortschreitenden Integration mit anderen betrieblichen Anwendungen gewinnt die Schnittstellentechnologie eines Modellierungswerkzeugs an Bedeutung. Hierbei können Kommunikations- und Softwareschnittstellen, Datenbankschnittstellen, Textschnittstellen, Grafikschnittstellen und Programmierschnittstellen unterschieden werden. Letztere ermöglichen die bedarfsorientierte Erweiterung des Funktionsumfangs eines Modellierungswerkzeugs durch den Endanwender.

Frage 4: Welche Beschreibungsmerkmale können zur Charakterisierung der Hauptkategorie „Technologie und Schnittstellen“ von GPM-Modellierungswerkzeugen herangezogen werden?

5. Methodik und Modellierung

Die Hauptkategorie „Methodik und Modellierung“ repräsentiert die für ein Modellierungswerkzeug charakteristischen Eigenschaften. Die Methodik umfasst folgende Beschreibungsmerkmale:

Methodenbezogene Unterkategorien

— **Methodenangebot:** Das vordefinierte Methodenangebot ist naturgemäß ein zentraler Funktionsumfang eines Modellierungswerkzeugs. Hierunter fallen allgemeingültige oder werkzeug-/anbieterspezifische Methoden. Diese können entsprechend ihrer Herkunft und Zielsetzung in strategische, prozessorientierte, organisationsorientierte, datenorientierte oder objektorientierte Methoden eingeteilt werden (z.B. UML-Diagramme, Ereignisgesteuerte Prozessketten, Petri-Netze).

— **Methodendefinition:** Neben vordefinierten Methoden ist die Funktionalität einer freien und endanwenderspezifischen Methodendefinition ein wichtiges Merkmal. Die Methodendefinition kann sich dabei auf Modelltyp-, Objekttyp- und Attributtyp-Ebene beziehen. Eine freie Methodendefinition durch den Endanwender und damit ein Eingriff in das Metamodell eines Werkzeugsystems steht aber oftmals in Konflikt zu einer vorgegebenen Repository-Struktur, welche im Regelfall anbieterspezifisch definiert und implementiert ist.

— **Methodentransformation:** Eine Methodentransformation beschreibt eine Funktion, mit der Methoden ineinander überführt werden können (z.B. die Überführung einer Ereignisgesteuerten Prozesskette in ein Petri-Netz oder ein UML-Aktivitätsdiagramm)

— **Methodenfilter:** Ein Methodenfilter ermöglicht es einem Endanwender, Methoden gezielt für einen Anwendungszusammenhang einzuschränken und quasi so zu „customizen“, dass nur Teilmengen der Modell-, Objekt- und Attributtypen an der Benutzerschnittstelle sichtbar sind. Somit können auch rollenbezogene Konzepte in den Prozess der Modellerstellung und -nutzung eingebracht werden.

Inhaltsbezogene Unterkategorien

— **Modellverwaltung:** Die Funktionen zur Modellverwaltung können in Explorer-, Sichten-, Ansichts- und Suchkonzepte strukturiert werden. Ein Explorerkonzept dient dazu, Modelle und Modellobjekte in einer baumartigen Dateistruktur abzulegen, aufzurufen, zu sortieren und zu selektieren. Diese Funktion ist in großen Modellstrukturen von zentraler Bedeutung. Das Sichtenkonzept unterstützt die perspektivische Strukturierung von Modellen (Organisationssicht, Prozesssicht, Datensicht, Ressourcensicht usw.). Ansichtskonzepte werden zur Visualisierung von Modell- und Objekteigenschaften, Modellübersichten (Orientierungsfenster), Tabellendarstellungen, Druck- und Präsentationsansichten, Ein- und Ausblendfunktionen und zur Definition eines Gültigkeitszeitraums für Modelle oder Modellobjekte benötigt. Suchkonzepte können auf Modell-, Objekt- und Attributebene eingesetzt werden.

— **Modellerstellung:** Die Bearbeitung von Modellen kann in grafischer und tabellarischer Form erfolgen. Je nach Zielsetzung und Sichtweise des Modellierungsprozesses kann es sinnvoll sein, beide Repräsentationsformen zu kombinieren. Zur Manipulation von Grafiken ist es wichtig, dem Modellierer leistungsfähige Manipulationsfunktionen bereitzustellen (Drag and Drop, Zwischenablage, Rasterfunktion, Hilfslinien, Lineal, Gitternetzlinien, Zoom, Gruppierung, Objektausrichtung usw.). Neben den Funktionen der unmittelbaren Modellerstellung ist die Einbindung von Fremdobjekten (Audio-, Video-, Text-, Grafik-, Hyperlinks), Freiformgrafiken und Freiformtexten wichtig. Hier werden oftmals wichtige Kontextinformationen abgelegt. Modellobjekte unterscheiden sich oftmals nur bezüglich einer Teilmenge ihrer Attributausprägungen. Mittels der Funktion der Attributvererbung wird der Modellierer bei der Übernahme von Attributausprägungen zwischen Modellobjekten unterstützt. Je nach Aufbau und Leistungsfähigkeit des Metamodells und der Datenhaltungstechnologie eines Modellierungswerkzeugs können verschiedene Kopierfunktionen angeboten

werden. Im Falle einer Definitionskopie wird ein neues Objekt mit komplementären Attributen und Attributausprägungen erzeugt. Eine Referenzkopie erzeugt nur eine neue grafische Zuordnung zu einer Modellgrafik, welche aber auf die gleichen Modellobjekteigenschaften im Repository verweist. Eine Manipulation an den Attributausprägungen einer Referenzkopie wirkt sich dann unmittelbar auf die Attributausprägungen aller zugehörigen grafischen Objektausprägungen aus. Modelle können, soweit dies methodisch unterstützt wird, entsprechend des gewünschten Blickwinkels verfeinert oder abstrahiert werden. Die Modellhierarchisierung ist daher eine Funktion, welche automatisiert den Entwickler bei der Erstellung und Verwaltung solcher Modellhierarchien unterstützt. Bei einer Modellgenerierung werden aus einer Datenbasis über die Referenzen gleicher Modellobjekte neue Modellgrafiken erzeugt. Diese Funktion ist beispielsweise wichtig, wenn entweder interaktiv sichtenbezogene Modellgrafiken erzeugt werden sollen oder vollautomatisch (Re-)Dokumentationen auf der Grundlage von Referenzmodellen generiert werden.

— **Modellkonsistenz:** Im Rahmen von Syntax- und Semantikprüfungen können modellbezogene Eigenschaften überprüft werden. Im Rahmen der Syntaxprüfung kommen entweder vordefinierte/standardisierte Prüfungen oder benutzerdefinierte Prüfprogramme zum Einsatz. Hierbei werden Modelle anhand methodenspezifischer Modellierungskonventionen auf ihre syntaktische Konformität überprüft (z.B. folgt in einem Prozessmodell auf ein Objekt vom Typ A stets ein Objekt vom Typ B und umgekehrt). Semantikprüfungen können in Analogie zu Syntaxprüfungen ebenfalls vordefiniert/standardisiert oder benutzerdefiniert erfolgen. Derartige Prüfungen (Verifikationen) setzen eine formale Beschreibung des dynamischen Systemverhaltens und eine modellspezifische Spezifikation zu prüfender Eigenschaften voraus (z.B. Erkennen von Deadlocks oder Livelocks in einem Prozessmodell).

Darstellungsbezogene Unterkategorien

— **Layout:** Das Layout eines Modells spielt für Kommunikations- und Präsentationszwecke eine herausragende Bedeutung. Der Funktionsumfang hierzu umfasst Funktionen zur Objektdarstellung (Größe, Farbe, Form, Schattierung, Fremdobjekt), Attributanzeige (Gruppierung, Baumstruktur, Grafikzuordnung), Definition von Schriftformaten, Formatvorlagen und der Arbeitsfläche (Skalierung, Farbe, Druckskalierung, Kopf- und Fußzeile). Im Rahmen der Layoutgenerierung wird eine vollautomatische relative Positionierung von Modellobjekten unterstützt. Die zugrunde liegenden Grafikalgorithmen können dabei in Abhängigkeit zu den eingesetzten Modellierungsmethoden konfiguriert werden (Layoutverfahren, Ausrichtung, Lage der Wurzelknoten usw.).

Verwaltungsbezogene Unterkategorien

— **Variantenmanagement:** Mit der zunehmenden Größe von Modellierungsvorhaben wächst die Bedeutung eines integrierten Variantenmanagements. Im Rahmen der Variantenerstellung wird auf Modell- und Objektebene die Definition und Verwaltung von Variantenbeziehungen unterstützt. Der Variantenvergleich und die Variantenkonsolidierung dienen der Zusammenführung von verteilten Modellinhalten.

— **Versionsmanagement:** Im Rahmen des Versionsmanagements können technologisch und historisch bedingte Versionen unterschieden werden. Das Versionsmanagement unterstützt dabei die Verwaltung und Konvertierung von Modellinhalten über verschiedene Versionen hinweg.

— **Vorgehensmodell:** Vorgehensmodelle können entweder vordefiniert/standardisiert sein und damit ein „festverdrahtetes“ Vorgehen im Modellierungsprozess erzwingen oder frei definiert werden.

— **Projektmanagement:** Funktionen zur Unterstützung des Projektmanagements können sich auf verschiedene Abstraktions- und Aggregationsstufen beziehen und sind oftmals durch eine mehr oder weniger integrierte Kopplung spezialisierter Fremdprodukte realisiert. Gegenstand ist die Verwaltung von Projektplänen, Aufgaben, Ressourcen, Kosteninformationen und CSCW-Funktionen.

Frage 5: Welche Beschreibungsmerkmale können zur Charakterisierung der Hauptkategorie „Methodik und Modellierung“ von GPM-Modellierungswerkzeugen herangezogen werden?

6. Anwendungen und Integration

Anwendungen bilden meist auch softwaretechnische (Teil-)Komponenten eines Modellierungswerkzeugs und bündeln einen bestimmten Funktionsumfang unter technischen oder betriebswirtschaftlichen Aspekten. Hierbei spielt die Integration von Fremdsystemen eine zunehmend bedeutendere Rolle.

Die Hauptkategorie „Anwendungen und Integration“ umfasst:

Präsentationsbezogene Unterkategorien

- **Navigation:** Funktionen zur Navigation beinhalten Konzepte zur Präsentation und Visualisierung von Modellen. Eine Online-Navigationskomponente ist ein spezieller Viewer/Reader, der auf der operativen Datenstruktur des Modellierungswerkzeugs aufsetzt und eine komfortable Lesefunktion beinhaltet. Eine Offline-Navigationskomponente basiert auf der Generierung einer eigenständigen Datenstruktur und erfolgt meist in einem webbasierten Dateiformat.
- **Animation:** Eine Animationsfunktion dient dazu, fallweise das dynamische Verhalten eines Modells abzubilden. Hierzu werden exemplarisch Instanziierungen des Modells erzeugt und durchgespielt. Man spricht in diesem Zusammenhang auch von einem Geschäftsvorfall (Business Case) oder einem Geschäftsszenario (Business Szenario). Funktionen zur Steuerung, Speicherung und Wiedergabe dienen der Kontrolle des Animationslaufes. Auf der Grundlage der Animationsläufe können dann auch Kennzahlen zu Kosten, Zeiten oder sonstigen Werten verdichtet und analysiert werden.

Auswertungsbezogene Unterkategorien

- **Analyse:** Anwendungen zur Analyse und Prozessverbesserung können entweder im Funktionsumfang eines Modellierungswerkzeugs vorgefertigt enthalten sein oder anwendungsorientiert erstellt werden. Die Ergebnisse eines Analysereports können dann grafisch oder tabellarisch aufbereitet werden. Typische Anwendungen sind Reports für Schwachstellenanalysen und Verbesserungspotenziale (z.B. Medienbrüche, Durchlaufzeiten usw.).
- **Kennzahlenermittlung:** Mittels kennzahlenbasierter Modelle können statische Anwendungen zur Kennzahlenermittlung eingesetzt werden. Ein Beispiel hierfür ist das Konzept zur Balanced Scorecard.
- **Prozesskostenrechnung:** Anwendungen für kostenrechnerische Verfahren können ebenfalls auf prozessorientierten Beschreibungsverfahren basieren. Hierbei werden auf der Grundlage von Anwendungsszenarien fallweise Kostenstrukturen erzeugt und bewertet (z.B. Kostentreiber).
- **Qualitätsmanagement:** Eine prozessorientierte Betrachtung qualitätsrelevanter Modellinformationen kann zur Planung, Steuerung und Kontrolle im Sinne eines integrierten Qualitätsmanagements genutzt werden (z.B. ISO 9000 ff. Zertifizierung).
- **Risikomanagement:** Modelle können als Grundlage für Funktionen des Risikomanagements eingesetzt werden. Hierzu müssen risikobezogene Kenngrössen modellspezfisch abgebildet werden (z.B. Prozessrisiken).
- **Simulation:** Die Simulation ermöglicht die dynamische Ausführung und Analyse von Geschäftsprozessmodellen. Hierzu können auf der Grundlage von Simulationsläufen Mengen, Zeiten und Kosten ermittelt und für Ressourcen Aussagen über Auslastungen und Engpässe ermittelt werden.

Ausführungsbezogene Unterkategorien

- **Datenbank (Re-)Engineering:** Der Import und Export von Schemainformationen eines Datenbanksystems ist ein weiteres Anwendungsgebiet für Modellierungswerkzeuge. Der Bezug zum Geschäftsprozessmanagement ergibt sich über die Beschreibung des Datenflusses.
- **Codegenerierung:** Anwendungen zur Codegenerierung können Bestandteil eines Modellierungswerkzeugs sein (z.B. für C++, Java, XML, SQL usw.). Diese Funktion wird aber meist über die Kopplung mit CASE-Werkzeugen realisiert.
- **Integration mit Fremdsystemen:** Die Integration mit Fremdsystemen eröffnet Modellierungswerkzeugen ein enormes Anwendungspotenzial. Hierzu können anwendungsbezogene Schnittstellen zu den Softwarekategorien Business Process Reengineering (BPR), Computer Aided Software Engineering (CASE), Computer Supported Cooperative Work (CSCW), Document Management Systems (DMS), E-Commerce, Enterprise Ressource Planning (ERP), Knowledge-Management-Systeme (KMS), Projektmanagement, Workflow Management (WFM) und Office-Anwendungen (z.B. Tabellenkalkulation, Textverarbeitung, Grafikverarbeitung usw.) bestehen.

7. Ausblick

Elektronischer Produktkatalog für GPM-Modellierungswerkzeuge

Das entwickelte Rahmenkonzept zur Evaluation von Modellierungswerkzeugen zum Geschäftsprozessmanagement wird derzeit sukzessiv mit den Daten marktgängiger Modellierungswerkzeuge gefüllt und dient als Grundlage für einen elektronischen Produktkatalog. Der elektronische Produktkatalog für Modellierungswerkzeuge soll Zielorganisationen bei einer umfassenden (Vor-)Auswahl relevanter Produkte unterstützen und eine benutzerindividuelle Gewichtung und Bewertung einzelner Merkmale ermöglichen. Auf der Grundlage der (Vor-)Auswahl können dann detaillierte Einzelanalysen und Einsatztests in der Zielorganisation erfolgen.

Literaturempfehlungen:

Bullinger,H.-J./Schreiner, P. (Hrsg.): Business Process Management Tools — Eine evaluierende Marktstudie über aktuelle Werkzeuge. Stuttgart 2001.

DIN (Hrsg.): Informationstechnik — Bewerten von Softwareprodukten — Qualitätsmerkmale und Leitfaden zu ihrer Verwendung. DIN 66272, Ausgabe:1994-10 (identisch mit ISO/IEC 9126:1991), Berlin 1994.

DIN (Hrsg.): Informationstechnik — Software-Erzeugnisse — Qualitätsanforderungen und Prüfbestimmungen, DIN ISO/IEC 12119, Ausgabe:1995-08 (identisch mit ISO/IEC 12119:1994), Berlin 1995.

Gartner Research: The BPA/M Market Gets a Boost From New Features. Gartner's Applications Development & Management Research Note, M-13-5295, 16.5.2001.

Gartner Research: The BPA Market Catches Another Major Updraft. Gartner's Application Development & Maintenance Research Note M-16-8153, 12.6.2002.

Herzwurm, G./Müller, U.: Kriterienkatalog zur Unterstützung der Auswahl von CASE-Tools. Studien zur Systementwicklung des Lehrstuhls für Wirtschaftsinformatik der Universität zu Köln, Band 1. Köln 1997.

Kirchner, L./Jung, J.: Ein Bezugsrahmen zur Evaluierung von UML-Modellierungswerkzeugen. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Nr. 26, Koblenz 2001.
URL: http://www.uni-koblenz.de/~iwi/publicfiles/Arbeitsberichte/ Nr26.pdf (02.12.2002)

Wettstein, T.: Vorschlag eines Kriterienkatalogs zur Evaluation eines WFMS. Arbeitspapier Universität Fribourg, Institut für Informatik, Fribourg 1999.
URL: http://www2-iiuf.unifr.ch/is/thomasw/WFSKrit.pdf (02.12.2002)

Die Beantwortung der Fragen erfolgt im WISU-Repetitorium.

Die Examensklausur aus der Wirtschaftsinformatik

Diese Aufgabe wurde von Prof. Dr. Rainer Thome an der Universität Würzburg im Prüfungstermin 2002/II für das Prüfungsfach Wirtschaftsinformatik in der Diplomprüfung der Studiengänge Betriebswirtschaft und Volkswirtschaft gestellt. Die maximale Bearbeitungszeit für die gesamte Klausur betrug 240 Minuten. Dabei standen in drei Teilgebieten je drei Aufgaben zur Wahl, von denen jeweils eine zu lösen war. Eine Aufgabe ist hier wiedergegeben. Bei der Erstellung der Musterlösung wirkten Dipl.-Kff. Katharina Angerhausen und Dipl.-Kfm. Christian Schneider mit.

Aufgabe: Informationssysteme

Bürokommunikation, Büroautomation, ERP, Vorgangssteuerung und Geschäftsprozessverbesserung können wie voneinander abgegrenzt werden?

I. Daran hätten Sie denken müssen:

Da die zu diskutierenden Begriffe in der Literatur und auch im Sprachgebrauch häufig unterschiedlich interpretiert werden, muss eine inhaltliche Abgrenzung erfolgen. Die **Bürokommunikation (BK)** fasst sämtliche Kommunikationsmittel und -wege auf Büroebene einer Unternehmung zusammen. Beispiele sind Telefon, Fax, Telex, E-Mail und Internet. Die **Büroautomation (BA)** umfasst organisatorische Maßnahmen zur effizienten Abwicklung und Gestaltung der Aufgaben auf Büroebene. Eine moderne Informationsverarbeitung führt zu einem effektiven Einsatz der BK. **Enterprise Resource Planning (ERP**) beinhaltet sämtliche unternehmensweiten Planungsaufgaben aus den Bereichen Personal, Anlage- und Umlaufvermögen, Zeitplanung usw. ERP-Systeme als Standardanwendungssoftware unterstützen die Planungsaufgaben durch eine integrierte Daten-

vor eine flächendeckende praxisorientierte Ausbildung verhindert wird. Für mehr als oberflächliche Kenntnisse kann es nicht ausreichen, Systeme nur von außen zu betrachten — man muss mit ihnen arbeiten.

Literaturempfehlungen:

Biethahn, J./Hummeltenberg, W./Schmidt, B./Stähly, P./Witte, T. (Hrsg.): Simulation als betriebliche Entscheidungshilfe. Heidelberg 1999.

Blumstengel, A./Kassanke, S./Suhl, L.: Praxisorientierte Lehre im Fachgebiet Operations Research unter Einsatz einer hypermedialen Lernumgebung. In: Wirtschaftsinformatik, 39. Jg. (1997), S. 555 - 562.

Fink, A./Schneidereit, G./Voß, S.: Grundlagen der Wirtschaftsinformatik. Heidelberg 2001.

Glover, F./Kelly, J.P./Laguna, M.: New Advances for Wedding Optimization and Simulation. In: Farrington, P.A./Nembhard, H.B./Sturrock, D.T./Evans, G.W. (Hrsg.): Proceedings of the 1999 Winter Simulation Conference (Vol. 1), IEEE, Piscataway 1999, S. 255 - 260.

Gutenschwager, K./Fauth, K.-A./Spieckermann, S./Voß, S.: Qualitätssicherung lagerlogistischer Steuerungssoftware durch Simulation. In: Informatik Spektrum, 23. Jg. (2000), S. 26 - 37.

Kostouriak, J./Gregor, M.: Simulation von Produktionssystemen. Berlin 1995.

Law, A.M./Kelton, W.D.: Simulation Modeling & Analysis. 3. Aufl., New York 2000.

Liem, S./Blecher, G./Gehr, F.: Simulation in der Geschäftsprozessoptimierung: Konzepte und Weiterentwicklungen. In: IM Information Management & Consulting 12, Sonderausgabe 1997, S. 64 - 68.

Page, E.H./Buss, A./Fishwick, P.A./Healy, K.J./Nance, R.E./Paul, R.J.: Web-based Simulation: Revolution or Evolution? In: ACM Transactions on Modeling and Computer Simulation, Vol. 10 (2000), S. 3 - 17.

Spieckermann, S./Voß, S./Wortmann, D.: Praxisorientierte Klassifikationsmerkmale für ereignisorientierte Simulationswerkzeuge. In: Zeitschrift für Planung, 8. Jg. (1997), S. 81 - 97.

Swain, J.J.: Power Tools for Visualization and Decision-Making. In: OR/MS Today, Vol. 28 (2001), S. 52 - 63.

Taylor, S./Robinson, S./Paul, R. (Hrsg.): Progress in Simulation Research — Special Issue. In: Journal of the Operational Research Society. Vol. 51 (2000), S. 383 - 507.

Die Beantwortung der Fragen erfolgt im WISU-Repetitorium.

Hauptstudium

Modellierung und Simulation von Geschäftsprozessen

Dipl. Wirt.-Inf. Harald Kühn / Prof. Dr. Dimitris Karagiannis, Wien

Die Modellierung von Geschäftsprozessen ist mittlerweile in vielen Unternehmen ein fester Bestandteil der Geschäfts- und Organisationsentwicklung. Die Simulation von Geschäftsprozessen ermöglicht darüber hinaus dynamische Auswertungen, um Zeit-, Kosten- und Kapazitätsanalysen durchzuführen. Die Dokumentation von Geschäftsprozessen und das prozessbasierte Management von Call-Centern sind beispielhafte Einsatzszenarien.

1. Einleitung

Kernelemente von Dienstleistungsunternehmen

Das zielgerichtete Management der Kernelemente eines Dienstleistungsunternehmens — der Produkte, der Geschäftsprozesse, der Organisationsstrukturen und der Informationstechnologie — zeichnet erfolgreiche Dienstleister aus (siehe Abb. 1). Ein Rahmenwerk zur Beantwortung der dabei immer wiederkehrenden Fragestellungen ist das Business Process Management Systems (BPMS)- Paradigma (vgl. Karagiannis et al. 1996).

Geschäftsprozess

Ein Geschäftsprozess ist eine Abfolge von Aktivitäten bzw. Subprozessen, die zur Erstellung eines Produktes von Akteuren durch Bearbeitung von Artefakten unter Zuhilfenahme von Ressourcen durchgeführt werden (vgl. Junginger 2000). Eine **Aktivität** ist eine elementare Arbeitseinheit. Immer wiederkehrende bzw. logisch zusammengehö-

rende Aktivitäten werden zu **Subprozessen** zusammengefasst. Die **Abfolge** beschreibt einen zeitlich-logischen Zusammenhang zwischen den Aktivitäten bzw. Subprozessen **(Kontrollfluss)** bezüglich eines Geschäftsprozesses. Als Produkte werden dabei Leistungen aller Art verstanden, d.h. nicht nur materielle Produkte, sondern insbesondere auch immaterielle Produkte wie Dienstleistungen. Ein **Akteur** führt die Aktivitäten durch. Ein Akteur kann sowohl eine Person als auch ein Informationssystem sein. Die von einem Akteur innerhalb eines Geschäftsprozesses bearbeiteten Einheiten wie Formulare, Informationen etc. werden Artefakte genannt. Der zeitlich-logische Zusammenhang zwischen den **Artefakten** und den Aktivitäten bzw. Subprozessen wird als **Informationsfluss** bezeichnet. Die **Ressourcen** sind die zur Bearbeitung der Artefakte bzw. zur Durchführung der Aktivitäten notwendigen Hilfsmittel.

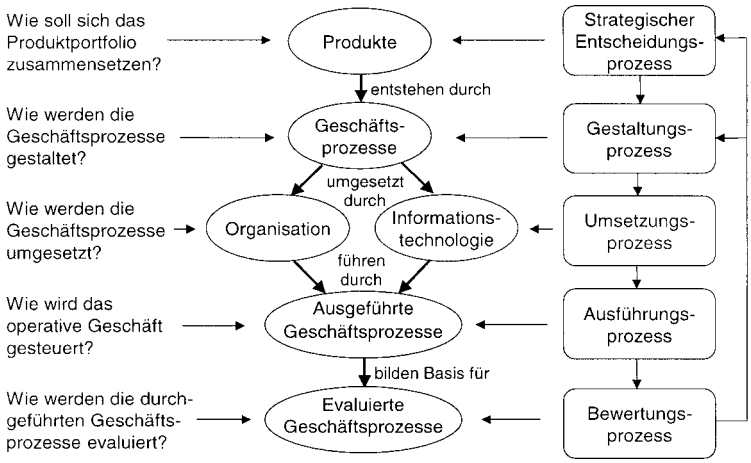


Abb. 1: Einordnung der Kernelemente eines Dienstleistungsunternehmens in das BPMS-Paradigma

Sichten auf Geschäftsprozessmodelle

Zur Beschreibung der für Geschäftsprozesse wichtigen Elemente werden häufig unterschiedliche Sichten betrachtet. In Curtis et al. 1992 werden zur Prozessmodellierung die vier Sichten „Funktional", „Dynamisch", „Organisatorisch" und „Inhaltlich" unterschieden. Für die Simulation von Geschäftsprozessen werden diese um die Sichten „Quantitativ" und „Zeitbezogen" ergänzt:

- **Funktionale Sicht:** Die funktionale Sicht beschreibt, aus welchen Aktivitäten und Subprozessen ein Geschäftsprozess aufgebaut ist.
- **Dynamische Sicht:** In der dynamischen Sicht werden die zeitlich-logischen Zusammenhänge beschrieben. Diese setzen sich aus dem Kontrollfluss und dem Informationsfluss zusammen.
- **Organisatorische Sicht:** Die organisatorische Sicht beschreibt die in den Geschäftsprozessen arbeitenden Akteure und die von den Akteuren eingesetzten Ressourcen.
- **Inhaltliche Sicht:** In der inhaltlichen Sicht werden die in den Geschäftsprozessen bearbeiteten Artefakte und die erstellten Produkte beschrieben.
- **Quantitative Sicht:** Die quantitative Sicht beschreibt die in einem Geschäftsprozess gebundenen Zeiten, Kosten, Wahrscheinlichkeiten und statistischen Verteilungen.
- **Zeitbezogene Sicht:** In der zeitbezogenen Sicht werden die zeitabhängigen Versionen und Varianten eines Geschäftsprozesses beschrieben.

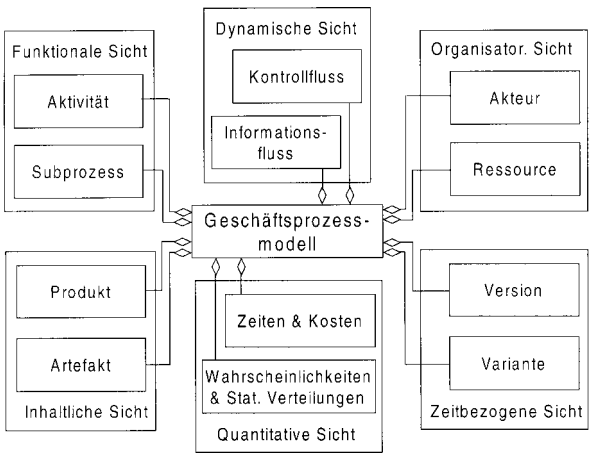


Abb. 2: Sichten auf Geschäftsprozessmodelle

Frage 1: Charakterisieren Sie den Zusammenhang der Kernelemente eines Dienstleistungsunternehmens im Kontext des Geschäftsprozessmanagements.

Modellierung und Simulation im Dienstleistungsbereich

Viele Simulationswerkzeuge zielen auf den Einsatz im Fertigungsbereich. Typische Einsatzszenarien sind dort die Simulation von Fertigungsstraßen oder von komplexen Produktionsprozessen. Die Modellierung und Simulation von Geschäftsprozessen im Dienstleistungsbereich stellt demgegenüber veränderte bzw. spezifische Anforderungen. Die folgende Liste gibt hierzu einen Überblick (vgl. auch Herbst et al. 1997):

- **Kundeneinbindung:** Der Kunde ist bei vielen Aktivitäten direkt oder indirekt beteiligt. Dieser externe Einfluss muss bei der Modellierung berücksichtigt werden.
- **Freiheitsgrade in der Aktivitätswahl:** In Dienstleistungsprozessen entscheiden die Akteure oft selbst über die nächste durchzuführende Aktivität. Darüber hinaus ist die Zuordnung von Aktivitäten zu Bearbeitern meist komplexer, beispielsweise bei umfangreichen Stellvertreterregelungen.
- **Kooperative Aktivitäten:** Viele Aktivitäten werden anstatt von einem einzelnen Akteur von einer Gruppe von Akteuren durchgeführt, beispielsweise gemeinsame Besprechungstermine, Workshops oder Präsentationen.
- **Unterbrechbarkeit von Aktivitäten:** Die Bearbeitung von Aktivitäten kann nicht einfach unterbrochen werden. Beispielsweise kann ein Mitarbeiter im Call-Center nicht einfach das momentane Telefonat beenden oder eine Krankenschwester im Krankenhaus die Behandlung eines Patienten einstellen, nur weil die jeweilige Arbeitsschicht zu Ende ist.
- **Ermittlung von Verantwortlichkeiten:** Beim Einsatz von warteschlangenbasierter Simulation muss eine präzise Zuteilung von Aktivitäten zu Akteuren erfolgen. Die ist im Fertigungsbereich einfacher, da Akteure meistens Maschinen mit klar definierten Tätigkeiten sind.
- **Variable Bearbeitungszeiten:** Die Bearbeitungszeiten von Aktivitäten sind im Fertigungsbereich nahezu stabil. Die Bearbeitungszeiten im Dienstleistungsbereich besitzen vielfach eine große Varianz basierend auf der momentanen Auslastung des jeweiligen Akteurs.
- **Trennung von Ablauf- und Aufbauorganisation:** Im Dienstleistungsbereich werden gleiche Geschäftsprozesse häufig in unterschiedlichen Lokationen durchgeführt, beispielsweise ein Kontoeröffnungsprozess einer Bank in unterschiedlichen Bankfilialen. Hierzu sollte die funktionale Sicht und die organisatorische Sicht getrennt modellier-, aber integriert simulierbar sein.
- **Einfachheit:** Im Dienstleistungsbereich ist die Anzahl von Mitarbeitern mit fundierten Simulationskenntnissen erfahrungsgemäß geringer als im Fertigungsbereich. Hierfür muss die Modellierung und Simulation deutlich einfacher handhabbar sein.

Frage 2: Beschreiben Sie notwendige Sichten für die Modellierung und Simulation von Geschäftsprozessen und skizzieren Sie deren Zusammenhänge.

2. Modellierung von Geschäftsprozessen

Modellierungsansätze

In der Wirtschaftsinformatik und den verwandten Disziplinen werden unterschiedliche Modellbegriffe eingesetzt. Im vorliegenden Zusammenhang werden Modelle sowohl als „Abbilder eines Realweltausschnitts“, beispielsweise ein Modell eines Geschäftsprozesses aus der Versicherungsbranche, als auch als „Konstruktionen zur Gestaltung der Realwelt“, beispielweise ein Workflow-Modell in der prozessorientierten Anwendungsentwicklung, verstanden.

2.1. Ziele einer Geschäftsprozessmodellierung

Ziele der Modellierung von Geschäftsprozessen

Die Erstellung von Geschäftsprozessmodellen sollte immer konkreten Zielen dienen. Hierzu gehören beispielsweise:

- Die **Dokumentation** von Prozesslandkarten (Wertschöpfungsketten) und den dazugehörenden Geschäftsprozessen. Die Dokumentation kann anschließend für eine Zertifizierung im Rahmen eines Qualitätsmanagements genutzt werden (ISO 9000:2000 usw.).
- Basierend auf Geschäftsprozessmodellen können **Mitarbeiterschulungen** durchgeführt werden. Dies ist insbesondere bei Unternehmen mit einer hohen Mitarbeiterfluktuation interessant.

— Die **Anpassung** der Geschäftsprozesse an sich ändernde Rahmenbedingungen beispielsweise bei Gesetzesänderungen oder die qualitative sowie quantitative **Optimierung** von bestehenden Abläufen.
— Die Definition von **fachlichen Vorgaben für die Anwendungsentwicklung** (Pflichtenhefte) basierend auf Soll-Geschäftsprozessen. Hierdurch kann die Qualität des zu erstellenden Informationssystems durch klar spezifizierte und kommunizierte Anforderungen deutlich erhöht werden.
— Der Aufbau von so genannten **„Process Warehouses"**, die u.a. Unterstützung bei der betriebswirtschaftlichen Auswertung operativer Daten und dem **Benchmarking** mit anderen Branchen bzw. Unternehmen bieten.

2.2. Abstraktionsebenen in der Geschäftsprozessmodellierung

Abstraktionsebenen

In der Geschäftsprozessmodellierung können verschiedene Abstraktionsebenen und Modelladressaten unterschieden werden.

Auf der **Ebene der Prozesslandkarten** wird eine Übersicht über alle bzw. die relevanten Geschäftsprozesse erstellt, die normalerweise keinen Kontroll- und Informationsfluss enthält und aus der „Vogelperspektive" erstellt wird. Die typischen Adressaten sind u.a. **Unternehmens- und Bereichsleiter und die jeweiligen Prozessverantwortlichen**.

Die **Ebene der Geschäftsprozesse** stellt die fachliche Verfeinerungsstufe der Prozesslandkarten dar. Dabei wird ein Prozess aus der Landkartenebene in ein oder mehrere Geschäftsprozessmodelle dekomponiert. Für die Landkarten- und Geschäftsprozessebene häufig eingesetzte Klassifikations- und Strukturierungskriterien sind Kern-, Management- und Unterstützungsprozesse oder die Unterteilung nach Produktgruppen und Produkten. Die typischen Adressaten sind u.a. **Prozessverantwortliche und Geschäftsprozessexperten**.

Die **Workflow-Ebene** spiegelt die fachlichen Geschäftsprozesse innerhalb der Informationssysteme wider. Workflows sind grundsätzlich strukturell den korrespondierenden Geschäftsprozessen ähnlich, fokussieren aber auf die informationstechnische Umsetzung (vgl. Karagiannis 1998). Im Gegensatz zu Geschäftsprozessmodellen, wo fachliche Beschreibungen, Zeiten, Kosten, Mengen etc. im Zentrum der Betrachtung stehen, beschreiben Workflow-Modelle die notwendigen Programmaufrufe, die benötigen Daten etc. Die typischen Adressaten sind u.a. **IT-Spezialisten**, **Systemarchitekten** und **Anwendungsentwickler**.

Den nächsten Verfeinerungsschritt stellt die **Mikro-Flow-Ebene** dar. Micro-Flows beschreiben den Ablauf innerhalb eines Workflow-Schritts, beispielsweise den Steuerungsfluss einer eingebundenen Applikation. Micro-Flows werden häufig mit Zustandsdiagrammen, Datenflussdiagrammen oder Petri-Netzen beschrieben. Die typischen Adressaten sind u.a. **Anwendungsentwickler** und **Programmierer**.

Übersicht der Ebenen

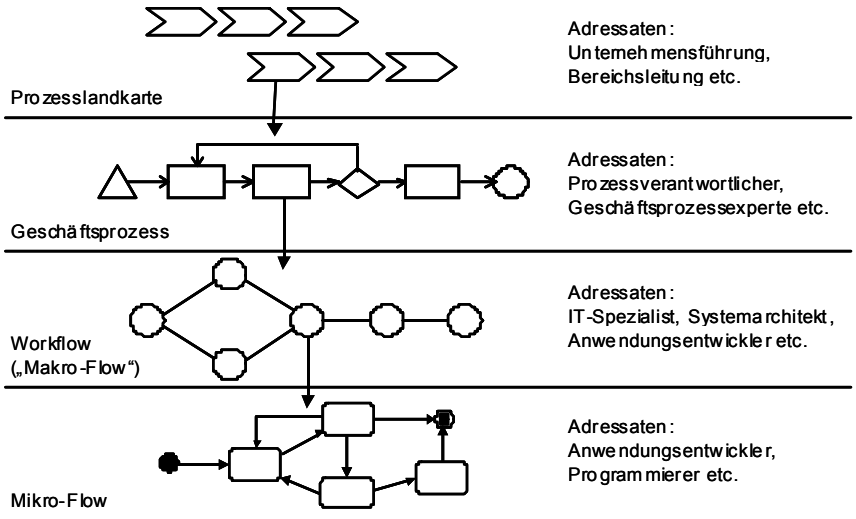


Abb. 3: Abstraktionsebenen in der Geschäftsprozessmodellierung

2.3. Modellierungssprachen für Geschäftsprozesse

Modellierungssprache und Metamodell

Ein Modell wird unter Einsatz einer Modellierungssprache erstellt. Eine Modellierungssprache besteht aus Modellierungselementen und Regeln, wie die Modellierungsele-

mente zur Modellbildung eingesetzt werden können. Dem sprachbasierten Metamodellbegriff folgend, wird unter einem Metamodell das Modell einer Modellierungssprache verstanden (vgl. Strahinger 1996, S. 24).

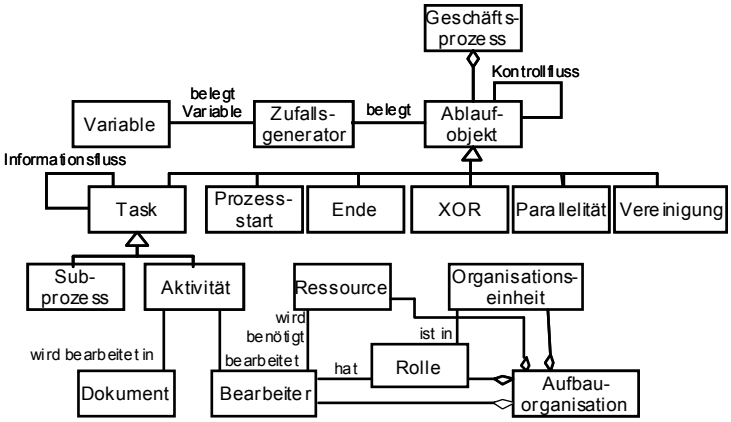


Abb. 4: Metamodellausschnitt der ADONIS-Sprache für Geschäftsprozessmodellierung

Es existieren mittlerweile eine Vielzahl von Sprachen zur Geschäftsprozessmodellierung. Im Folgenden werden das Aktivitätsdiagramm der Unified Modeling Language (UML), die ereignisgesteuerte Prozesskette (EPK), die Line of Visibility Engineering Methodology (LOVEM) und die ADONIS-Sprache für die Geschäftsprozessmodellierung kurz erläutert. Für eine ausführliche Beschreibung sei auf die jeweilige Literaturangabe verwiesen.

Aktivitätsdiagramm der UML

Die **UML** ist eine Modellierungssprache für die Spezifikation objektorientierter Software-Systeme und besteht aus neun Modellarten. Das zentrale Element des UML-Aktivitätsdiagramms ist die Aktivität (vgl. Kühn/Junginger 1999; OMG 1999). Mit den Modellierungselementen des Aktivitätsdiagramms kann die funktionale, dynamische, organisatorische und inhaltliche Sicht beschrieben werden.

Ereignisgesteuerte Prozesskette (EPK)

Die Basiselemente einer **EPK** sind Ereignisse und Funktionen (Keller et al. 1992). Ereignisse lösen Funktionen aus, Funktionen produzieren Ereignisse. Ereignisse und Funktionen können über die logischen Operatoren „Exklusives ODER", „UND" und „ODER" verknüpft werden. Es kann die funktionale, dynamische, organisatorische und inhaltliche Sicht beschrieben werden.

Line of Visibility Engineering Methodology (LOVEM)

In **LOVEM** werden die Architektur-, die logische, die physische und die Job-Ebene unterschieden (vgl. IBM 1999). Im Zentrum der Modellierung steht der Kunde. Geschäftsprozesse werden immer „vom Kunden zum Kunden" als Prozess- bzw. Aktivitätsketten, ähnlich den Aktivitätsdiagrammen in UML, abgebildet. Die Line of Visibility beschreibt die Schnittstelle zwischen Kunde und Unternehmen. Es kann die funktionale, dynamische, organisatorische und inhaltliche Sicht beschrieben werden.

ADONIS-Sprache für Geschäftsprozess-modellierung

Die **ADONIS-Sprache** für Geschäftsprozessmodellierung enthält u.a. „Prozesslandkarten" und „Geschäftsprozessmodelle" (vgl. BOC 2001). Die zentralen Modellierungselemente sind „Prozess" und „Aktivität". Diese können über die logischen Operatoren „Entscheidung", „Parallelität" und „Vereinigung" verknüpft werden. Es kann die funktionale, dynamische, organisatorische und inhaltliche Sicht beschrieben werden.

Frage 3: Beschreiben Sie Abstraktionsebenen in der Geschäftsprozessmodellierung. Gehen Sie dabei insbesondere auf die Zusammenhänge der Ebenen und auf die Modelladressaten ein.

3. Simulation von Geschäftsprozessen

Rechnerische und simulationsbasierte Auswertung

Die quantitative Auswertung von Geschäftsprozessmodellen kann auf unterschiedliche Arten erfolgen. Eine Möglichkeit ist die Auswertung basierend auf analytischen Verfahren, d.h. eine **rechnerische Auswertung**. Die Basis bilden dabei Wahrscheinlichkeiten von Prozesspfaden, daraus ermittelte Aktivitätshäufigkeiten und bei den Aktivitäten hinterlegte Zeit- und Kostenattribute. Eine rechnerische Auswertung kann jedoch bei komplexen Geschäftsprozessmodellen an die Grenzen der Berechenbarkeit stoßen. Beispielsweise kann die Rechenzeit für die Ergebnisermittlung exponentiell mit der Anzahl der in einem Geschäftsprozessmodell enthaltenen Entscheidungen steigen. Ein Lösungsansatz hierfür ist der Einsatz von **Simulationsverfahren**. Dabei wird ein Ge-

schäftsprozess ausreichend oft „ausgeführt“ und die einzelnen Ausführungsergebnisse mit ihrer jeweiligen Wahrscheinlichkeit gewichtet. Je größer die Anzahl der durchgeführten Simulationsläufe ist, desto exakter werden die Ergebnisse.

3.1. Grundbegriffe

Quantitative Kriterien

Bei der Simulation von Geschäftsprozessen kann eine Vielzahl von quantitativen Kriterien betrachtet werden. Abb. 5 gibt hierzu eine Übersicht. Die Betrachtung von qualitativen Kriterien wird hier nicht näher beschrieben.

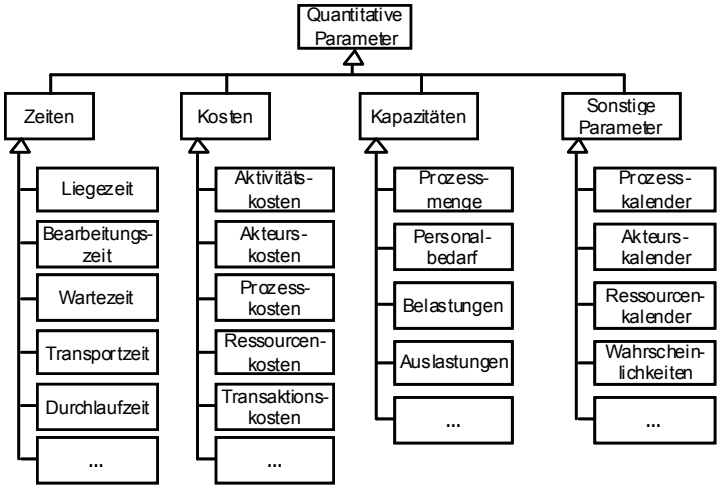


Abb. 5: Parameterübersicht für die Simulation von Geschäftsprozessen

Zeiten

Häufig ausgewertete Zeitparameter sind (vgl. BOC 2001, S. 369 f.):

- **Durchlaufzeit:** Die Durchlaufzeit eines Geschäftsprozesses gibt die Zeit an, die durchschnittlich vom Start bis zur Beendigung des Geschäftsprozesses verstreicht.
- **Bearbeitungszeit:** Die Bearbeitungszeit eines Geschäftsprozesses gibt die Zeit an, die durchschnittlich für die Bearbeitung der Aktivitäten anfällt.
- **Wartezeit:** Die Wartezeit eines Geschäftsprozesses gibt die Zeit an, die durchschnittlich vor der Bearbeitung bzw. bei der Unterbrechung der Aktivitäten anfällt.
- **Liegezeit:** Die Liegezeit eines Geschäftsprozesse gibt die Zeit an, welche die Aktivitäten durchschnittlich bei den Akteuren nach der Bearbeitung „liegen bleiben“.
- **Transportzeit:** Die Transportzeit eines Geschäftsprozesses gibt die Zeit an, die durchschnittlich für den Transport zwischen den Aktivitäten anfällt.

Durchlaufzeit und Parallelitäten

Die Durchlaufzeit eines Geschäftsprozesses wird durch parallel ausgeführte Aktivitäten beeinflusst. Die Durchlaufzeit beträgt dann nicht mehr die Summe der einzelnen Zeitparameter, sondern innerhalb einer Parallelität geht der jeweilige Maximalwert in die Berechnung ein. In Abb. 6 beträgt die Durchlaufzeit demnach nicht 20 Minuten, sondern 15 Minuten. Diese setzt sich aus der Bearbeitungszeit von A1 und dem Maximum der Bearbeitungszeiten von A2 und A3 zusammen.

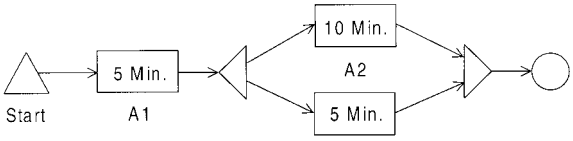


Abb. 6: Auswirkung von Parallelitäten auf die Durchlaufzeit

3.2. Vorgehensweise

Vorgehensweise zur Geschäftsprozess-simulation

Bei der Erstellung, der Auswertung und der Validierung eines Simulationsmodells können grundsätzlich folgende Schritte unterschieden werden:

- In der **Zieldefinition** werden der zu untersuchende Problembereich und die hierfür zu ermittelnden Ergebnisgrößen festgelegt. Basierend auf den Ergebnisgrößen werden die für die Simulation notwendigen Input-Parameter definiert.
- Danach erfolgt die **Erhebung** der als notwendig definierten Input-Parameter und die Erstellung der Geschäftsprozessmodelle. Dabei ist auf eine ausreichende **Qualitätssicherung** der erhobenen Informationen und Geschäftsprozesse zu achten. Dies wird beispielweise in Form von moderierten Workshops, Interviews und Reviews durchgeführt.

— Die **Auswertung** der erhobenen und qualitätsgesicherten Geschäftsprozessmodelle wird anschließend mit Simulationsverfahren durchgeführt. Die Auswertungsmöglichkeiten reichen dabei von reinen Prozessanalysen bis hin zu komplexen Auswertungen unter Nutzung von unterschiedlichen organisatorischen, quantitativen und zeitbezogenen Sichten. Beispielsweise kann ein Geschäftsprozess in unterschiedlichen Aufbauorganisationen in unterschiedlichen Varianten simuliert werden.

Entscheidungsgrößen und diagnostische Größen

— Anschließend erfolgt die **Überprüfung und Interpretation** der ermittelten Ergebnisse auf Plausibilität und Realitätsnähe. Dabei kann zwischen Entscheidungsgrößen und diagnostischen Größen unterschieden werden. Entscheidungsgrößen orientieren sich an den in der Zieldefinition festgelegten Ergebnisgrößen und erlauben Aussagen wie „besser" oder „schlechter". Entscheidungsgrößen sind beispielsweise die Bearbeitungs- und Durchlaufzeit eines Geschäftsprozesses oder der benötigte Personalbedarf zur Erledigung einer bestimmten Prozessmenge. Diagnostische Größen sind geeignet um zu erklären, warum sich ein Modell in einer bestimmten Art und Weise verhalten hat. Die Wartezeit kann beispielsweise eine diagnostische Größe sein, um das Durchlaufzeitverhalten eines Geschäftsprozesses zu erklären.

Anhand der Überprüfung und Interpretation kann es notwendig sein, dass das erstellte Geschäftsprozessmodell oder die Input-Parameter adaptiert werden müssen. Dann beginnt die beschriebene Vorgehensweise von vorne.

3.3. Simulation von Geschäftsprozessen ohne „Organisatorische Sicht"

Pfadanalyse

Bei der simulationsbasierten Auswertung von Geschäftsprozessen ohne Berücksichtigung der zugrundeliegenden Aufbauorganisation spricht man von einer Pfadanalyse. Wesentliche **Input-Größen** sind dabei ein Geschäftsprozessmodell und die dazugehörenden Subprozessmodelle, die Hinterlegung des Kontrollflusses mit entsprechenden Pfadwahrscheinlichkeiten und aktivitätsbezogene Zeiten- und Kostenfaktoren. Die Output-Größen lassen sich in folgende Ergebnistypen klassifizieren:

— **Geschäftsprozessergebnisse** können z.B. Beispielsweise die durchschnittliche Durchlaufzeit oder die durchschnittliche Bearbeitungszeit eines Geschäftsprozesses.
— **Pfadergebnisse** sind z.B. die durchschnittliche Durchlaufzeit oder die durchschnittliche Wahrscheinlichkeit eines konkreten Pfades. Ein weiteres Ergebnis sind nicht durchlaufene („tote") Pfade.
— **Aktivitätsergebnisse** sind z.B. die durchschnittliche Bearbeitungszeit, falls die Input-Größe mit einer stochastischen Verteilung beschrieben wurde, oder die Aktivitätshäufigkeit bezogen auf einen Prozessdurchlauf.

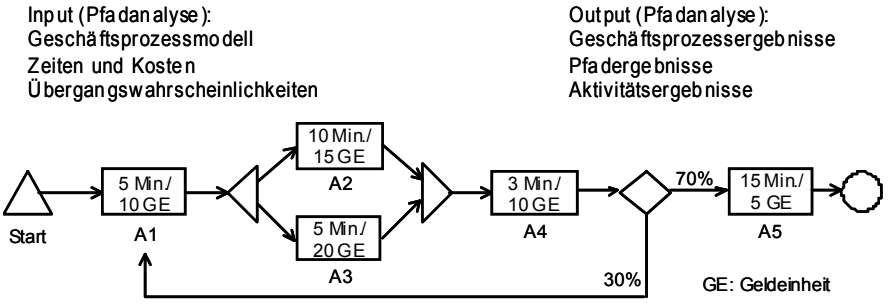


Abb. 7: Geschäftsprozesssimulation ohne „Organisatorische Sicht"

3.4. Simulation von Geschäftsprozessen mit „Organisatorischer Sicht"

Bei der Simulation von Geschäftsprozessen unter Berücksichtigung der Aufbauorganisation können grundsätzlich zwei Ansätze unterschieden werden.

Belastungsanalyse

— Bei der **kapazitätsorientierten Betrachtung** (Belastungsanalyse) sind zusätzlich zu den Input-Größen der Pfadanalyse und den Elementen der Aufbauorganisation die **Mengengerüste** der jeweiligen Geschäftsprozesse notwendig. Ein Mengengerüst beschreibt dabei, in welcher Periode ein Geschäftsprozess mit welcher Menge auftritt. Die Simulation erfolgt nicht anhand einer Zeitachse. Die Belastungsanalyse beantwortet u.a. die Frage: „Wie viele Akteure und wie viele Ressourcen werden benötigt, um die über die Mengengerüste beschriebenen Prozessmengen bearbeiten zu können?" Ein Einsatzschwerpunkt der Belastungsanalyse ist die Personalbedarfsrechnung.

Auslastungsanalyse

— In der **zeitbezogenen Betrachtung** (Auslastungsanalyse) wird anhand einer Zeitachse simuliert. Zusätzlich zu den Input-Größen der Pfadanalyse und den Elementen

der Aufbauorganisation besitzen die Geschäftsprozesse, die Akteure und die Ressourcen so genannte Kalender. Ein **Prozesskalender** beschreibt mittels statistischer Verteilungen die Anstoßintervalle für das Starten des korrespondierenden Geschäftsprozesses. Ein **Akteurskalender** beschreibt die Anwesenheitszeiten eines Akteurs, in denen einem Akteur Aktivitäten aus den Geschäftsprozessen zur Bearbeitung zugeteilt werden können. Ein **Ressourcenkalender** beschreibt die Zeit, die eine Ressource zur Verfügung steht, um durch die Akteure genutzt werden zu können. Eine Hauptaufgabe der Auslastungsanalyse ist die Ermittlung von Wartezeiten, die bei der Bearbeitung von Aktivitäten entstehen, falls nicht genügend Akteure bzw. Ressourcen zur Verfügung stehen. Die Auslastungsanalyse beantwortet die Frage: „Wie verhalten sich die Prozessdurchlaufzeiten und wo sind die Engpässe in der Prozessbearbeitung?“ Ein Einsatzschwerpunkt der Auslastungsanalyse ist die Szenariobewertung für Ist-Soll-Vergleiche.

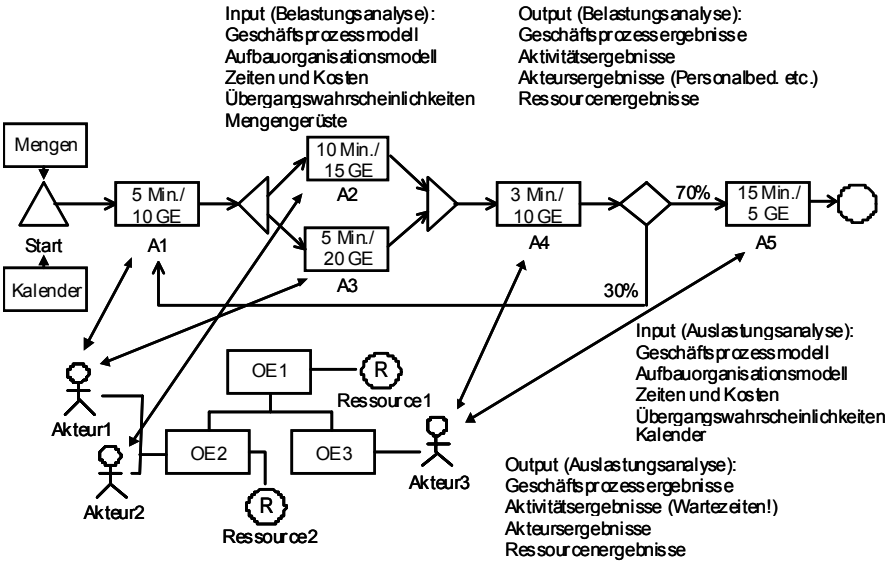


Abb. 8: Geschäftsprozesssimulation mit „Organisatorischer Sicht“

3.5. Animation

Animation

Zur Visualisierung des dynamischen Verhaltens von Simulationsmodellen können **Animationstechniken** eingesetzt werden. Folgende Übersicht zeigt einige Möglichkeiten, wie die Animation in der Simulation von Geschäftsprozessen eingesetzt werden kann:

- Durch die **Animation des Kontrollflusses** wird der Status der Prozessabarbeitung visualisiert. Bei Einbindung zusätzlicher Informationsquellen in die Animation, beispielsweise Videosequenzen, kann dies auch für Schulungszwecke eingesetzt werden.
- Die **Animation des Informationsflusses** unterstützt bei der Analyse der Kommunikationsstrukturen.
- Eine **Animation der Akteure und der Ressourcen**, beispielsweise über Wartestapel, gibt Aufschluss über potenzielle Engpässe in der Aktivitätsbearbeitung.
- Eine **Animation der Ergebnisentwicklung** kann Hilfestellung bei der Erstellung von Prognosen liefern.

Frage 4: Skizzieren Sie unterschiedliche Möglichkeiten der simulationsbasierten Auswertung von Geschäftsprozessen und erläutern Sie den jeweiligen Einsatzschwerpunkt.

4. Einsatzbereiche

Es werden nun zwei Einsatzbereiche der Modellierung und Simulation von Geschäftsprozessen skizziert. Der erste Einsatzbereich — die Dokumentation von Geschäftsprozessen — hat den Schwerpunkt auf der Modellierung. Im zweiten Einsatzbereich — dem prozessbasierten Management von Call-Centern — steht die Simulation im Vordergrund.

4.1. Dokumentation von Geschäftsprozessen

Prozessdokumentation

Die Geschäftsprozessmodellierung wird momentan hauptsächlich im Rahmen der Erstellung einer umfassenden **Dokumentation der Unternehmensprozesse** eingesetzt.

Die **Erhebung und Modellierung** der Geschäftsprozesse wird dabei entweder durch die Fachbereiche selbst oder mit Unterstützung durch interne Mitarbeiter aus Abteilungen wie „Organisation", „Methoden und Verfahren" oder „Inhouse-Consulting" sowie auch durch externe Berater durchgeführt. Der Fokus liegt dabei häufig auf den Ebenen „Prozesslandkarte" und „Geschäftsprozess".

Zur Verkürzung der Erhebungszeit können hierbei auch **Referenzmodelle** zum Einsatz kommen, die beispielsweise branchenspezifisch erworben werden können. Nach der **Qualitätssicherung und der Abnahme der modellierten Geschäftsprozesse** durch die verantwortlichen Abteilungen und Unternehmensbereiche werden die Prozessdokumentationen **an die Mitarbeiter verteilt**. Dies kann sowohl in Form von traditionellen Papierdokumentationen als auch in elektronischer Form für gängige Textverarbeitungs- und Publishing-Systeme oder als Intranet-Dokumentation erfolgen. Insbesondere durch den Einsatz eines Intranets können die Prozessdokumentationen rasch kommuniziert und aktuell gehalten werden.

Für die Erstellung einer Prozessdokumentation werden folgende **Hauptziele** genannt:

- **Steigerung der Transparenz** der Geschäftsprozesse und hierdurch Aufdeckung potenzieller Schwachstellen
- Ausrichtung der Geschäftsprozesse auf interne und externe Kunden und **Erhöhung der Kundenzufriedenheit**
- Etablieren eines **prozessbasierten Qualitätsmanagements**; dies kann bei Bedarf sogar zur Zertifizierung nach unterschiedlichen Qualitätsstandards führen.
- Erhalt bzw. Steigerung der **Wettbewerbsfähigkeit** durch Möglichkeiten des Benchmarkings basierend auf den Geschäftsprozessmodellen

4.2. Prozessbasiertes Call-Center-Management

Prozessbasiertes Call-Center-Management

Der Aufgabenbereich von Call-Centern entwickelt sich immer mehr von der reinen Telefonzentrale hin zur **Kommunikationszentrale**. Dabei ist das Telefon nicht mehr das einzige Kommunikationsmedium, sondern es müssen auch Post, Fax, E-Mail, Internet und mobile Geräte integriert werden. Hierdurch vervielfältigen sich die notwendigen Geschäftsprozesse, Aufgaben und die Rollen der beteiligten Akteure.

Eine Hauptaufgabe im Management von Call-Centern ist die **Planung von Mitarbeiterkapazitäten** in Abhängigkeit von definierten **Service-Levels**, d.h. zugesicherten Qualitäts- bzw. Bearbeitungsstandards. Ein oft angewandter Service-Level ist die „80:20-Regel", d.h., 80% der eingehenden Anrufe müssen innerhalb von 20 Sekunden angenommen werden. Herkömmliche rechnerische Verfahren für die Kapazitätsplanung in Call-Centern basieren auf „Erlang B"- oder „Erlang C"-Ansätzen. Diese machen einige vereinfachende Annahmen wie die Gleichbehandlung aller Anrufer, die Bearbeitung eines Anrufs durch genau eine Rolle bzw. einen Mitarbeiter oder dass die Anzahl abgebrochener Anrufe unabhängig von der Dauer in der Warteschleife ist.

Der Einsatz von g**eschäftsprozessbasierten Simulationsverfahren** in der Kapazitätsplanung im Call-Center bringt u.a. folgende **Vorteile**:

- **Unterschiedliche Anrufe** können auch unterschiedlich behandelt werden, beispielsweise in Form variierender Bearbeitungszeiten.
- Die **Einbeziehung des operativen Anruferverhaltens** aus der ACD-Anlage (Automatic Call Distribution) in die Belegung von Prozesskalendern erhöht die Realitätstreue.
- **Rollenwechsel in der Anfragebearbeitung** werden ermöglicht. Dies resultiert in realistischer Kapazitätsplanungen und in eine Prozessbetrachtung vom Kundenkontakt bis zur abschließenden Fallbearbeitung.
- Das **Abbruchverhalten der Anrufer** wird in Abhängigkeit von Wartezeiten in der Anruferwarteschlange ausgewertet.

Frage 5: Erläutern Sie zwei Einsatzszenarien für die Modellierung und Simulation von Geschäftsprozessen.

5. Schlussbetrachtungen

Prozesse sind Unternehmens-Know-How

Die Modellierung von Geschäftsprozessen ist in vielen Unternehmen ein fester Bestandteil in der Geschäfts- und Organisationsentwicklung. Dabei werden Geschäftsprozessmodelle verstärkt als ein Element im Rahmen des **Wissensmanagements** eingesetzt.

Die anhaltende „Diffusion“ von Workflow-Technologie in Plattformen zur Anwendungsentwicklung lässt ebenfalls auf eine weitere Verbreitung der Geschäftsprozessorientierung schließen (vgl. Junginger 2000). Simulation ist insbesondere hilfreich bei der quantitativen Auswertung von komplexen Modellen. Auch besitzen simulationsfähige Geschäftsprozessmodelle eine höhere Qualität bezüglich der Ausführbarkeit, was vor allem beim Einsatz von Geschäftsprozessmodellen als **Vorgaben für eine prozessorientierte Anwendungsentwicklung** zur Steigerung der Softwarequalität führen kann. Der Aufwand für die Erhebung von quantitativen Informationen für die Simulation darf nicht außer Acht gelassen werden. Hier erwarten wir durch die **Rückführung operativer Daten** in modellauswertende Werkzeuge, beispielweise aus Workflow-Protokollen oder Standardsoftware-Paketen, eine deutliche Aufwandsreduktion.

Literaturempfehlungen:

BOC: ADONIS Benutzerhandbuch Version 3.5. BOC Information Technologies Consulting GmbH, http://www.boc-eu.com, 2001.

Curtis, B./Kellner, M.I./Over, J.: Process Modeling. In: Communications of the ACM, 35. Jg. (1992), No. 9, S. 75 - 90.

Herbst, J./Junginger, S./Kühn, H.: Simulation in Financial Services with the Business Process Management System ADONIS. In: Hahn, W./Lehmann, A. (Hrsg.): Proceedings of the 9th European Simulation Symposium (ESS‘97), Society for Computer Simulation (SCS), 1997, S. 491 - 495.

IBM: LOVEM — Line of Visibility Engineering Methodology — User‘s Guide, Version 3. 3. Aufl., IBM Canada Ltd., September 1999.

Junginger, S.: Modellierung von Geschäftsprozessen — State-of-the-Art, neuere Entwicklungen und Forschungspotenziale. BPMS-Report, Universität Wien, Institut für Informatik und Wirtschaftsinformatik, Abteilung Knowledge Engineering, Juni 2000.

Karagiannis, D.: Einsatz von Workflow-Technologien zur Umsetzung von Geschäftsprozessen. In: Moormann, J./Fischer, T. (Hrsg.): Informationstechnologie in Banken. Wiesbaden 1998, S. 385 - 403.

Karagiannis, D./Junginger, S./Strobl, R.: Business Process Management System Concepts. In: Scholz-Reiter, B./Stickel, E. (Hrsg.): Business Process Modelling. Berlin et al. 1996, S. 81 -1 06.

Keller, G./Nüttgens, M./Scheer, A.-W.: Semantische Prozessmodellierung auf der Grundlage „Ereignisgesteuerter Prozessketten (EPK)“. Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlandes, Heft 89, 1992.

Kühn, H./Junginger, S.: An Approach to Use UML for Business Process Modeling and Simulation in ADONIS. In: Szczerbicka, H. (Hrsg.): Proceedings of the 13th European Simulation Multiconference (ESM‘99) — Volume I, Society for Computer Simulation (SCS), 1999, S. 634 - 639.

OMG: Unified Modeling Language Specification 1.3, 1999. http://www.omg.org, Abruf am 15.06.2001.

Strahinger, S.: Metamodellierung als Instrument des Methodenvergleichs. Eine Evaluierung am Beispiel objektorientierter Analysemethoden. Aachen 1996.

Die Beantwortung der Fragen erfolgt im WISU-Repetitorium.

Hauptstudium

Geschäftsprozessmodellierung mit der ereignisgesteuerten Prozesskette

Prof. Dr. Dr. h.c. mult. August-Wilhelm Scheer /
Dipl.-Kfm. Oliver Thomas, Saarbrücken

Geschäftsprozessmodelle dienen der Kommunikation zwischen den Mitarbeitern einer Organisation und den Beratern oder Entwicklern von Anwendungssystemen. Ihre semiformale Beschreibung vermeidet sowohl die Unschärfe der natürlichen Sprache als auch die für viele Probleme unbrauchbare mathematische Formulierung. Dabei haben sich grafische Darstellungsformen durchgesetzt. Mit der ereignisgesteuerten Prozesskette (EPK) wird eine Modellierungssprache zur Konstruktion von Prozessmodellen vorgestellt, die seit mehr als einem Jahrzehnt breite Akzeptanz in Wissenschaft und Praxis findet.

1. Geschäftsprozessmanagement und -modellierung

Aufbauorganisation

In der Organisationslehre stand lange Zeit die Aufbauorganisation im Mittelpunkt. Diese Betrachtung lediglich zeitlich unabhängiger, statischer Regelungen, wie Hierarchien und Unternehmenstopologien, erfasst nur die Kommunikationsbeziehungen zwischen den Unternehmensteilen. Bis in die neunziger Jahre gestaltete sich die Koordination über künstlich erzeugte „Abteilungsmauern“ hinweg als äußerst schwierig. Lediglich das persönliche Engagement und die Abstimmung der Mitarbeiter untereinander konnten die auf diesem Wege erzeugte Sichtweise isolierter betrieblicher Aktivitäten ausgleichen. Die Automatisierung interner Verfahren, indem z.B. Informationen nicht mehr papierbasiert, sondern elektronisch übermittelt wurden, blieb häufig das einzige Verbesserungspotenzial.

Ablauforganisation und Geschäftsprozesse

Erst gegen Mitte der letzten Dekade veränderten Begriffe wie „Continuous Process Improvement“ oder „Business Process Reengineering“ die Sichtweise. Die Ablauforganisation, d.h. das zeitlich-logische, dynamische Verhalten von Vorgängen zur Aufgabenerfüllung der Unternehmen, rückte in den Vordergrund. Man versuchte, effiziente Automationsmodelle unternehmensweit zu implementieren. Die Kommunikation zwischen den Abteilungen und die daraus resultierende Orientierung an der Logik von Geschäftsprozessen rückten in den Mittelpunkt. Ein **Geschäftsprozess** ist eine zusammengehörende Abfolge von Verrichtungen zum Zweck der Leistungserstellung. Ausgang und Ergebnis eines Geschäftsprozesses ist eine Leistung, die von einem internen oder externen Kunden angefordert und abgenommen wird.

Geschäftsprozessmanagement und -modellierung

Ziel von Business-Engineering-Projekten ist die Gestaltung der Geschäftsprozesse und die Analyse der Anforderungen an deren IT-Unterstützung unter Berücksichtigung von Unternehmensstrategien. Die Gestaltung der Geschäftsprozesse muss einem umfassenden Ansatz folgen, der sowohl die Planung und Kontrolle als auch die Steuerung, d.h. das Management der betrieblichen Abläufe umfasst. Zur Unterstützung eines systematischen Vorgehens bei der Prozessgestaltung hat sich die Modellierung als hilfreich erwiesen. Modellierungssprachen, wie die ereignisgesteuerte Prozesskette (EPK), dienen als operationalisierter Ansatz zur Modellkonstruktion. Softwarewerkzeuge zur Geschäftsprozessmodellierung, wie das ARIS-Toolset der IDS Scheer AG (2003), können den Business Engineer durch Systemkomponenten zur Erhebung, Erstellung, Analyse und Simulation von Geschäftsprozessmodellen unterstützen.

Frage 1: Was versteht man unter einem Geschäftsprozess?

2. Einordnung und Historie der EPK

Prozessmodellierungssprachen

Seit Ende der siebziger Jahre wurde eine Vielzahl von Modellierungssprachen zur Beschreibung von Geschäftsprozessen entwickelt, z.B.

- Flussdiagramme in Form von Struktogrammen, Programmablauf- oder Datenflussplänen,
- Datenflussdiagramme im Rahmen der Structured Analysis (SA)-Methode,
- der Netzplan,
- das Vorgangskettendiagramm (VKD),
- das Petri-Netz,

— die ereignisgesteuerte Prozesskette (EPK) sowie
— Ansätze zur Prozessmodellierung bei objektorientierten Modellierungsmethoden.

Verbreitung der EPK im deutschsprachigen Raum

Zur Konstruktion von Geschäftsprozessmodellen auf fachkonzeptioneller Ebene hat sich aufgrund ihrer Anwendungsorientierung und umfassenden Werkzeugunterstützung insbesondere im deutschsprachigen Raum die EPK etabliert (vgl. Nüttgens/Rump 2002, S. 64). Sie wurde am Institut für Wirtschaftsinformatik (IWi), Saarbrücken, in Zusammenarbeit mit der SAP AG entwickelt (vgl. Keller/Nüttgens/Scheer 1992) und ist Bestandteil des **ARIS-Toolset** der IDS Scheer AG (2003) sowie des Business Engineering und Customizing des SAP R/3-Systems (vgl. Keller/Teufel 1999). Sie gehört neben dem VKD zu den wichtigsten an diesem Institut entwickelten Prozessbeschreibungssprachen und gilt als zentrale Modellierungssprache der Architektur integrierter Informationssysteme (ARIS) (vgl. Scheer 2001; 2002).

GI-Arbeitskreis zur EPK

Da die EPK in Forschung und Praxis als Beschreibungssprache für betriebliche Abläufe weit verbreitet ist, gründeten Nüttgens und Rump 2002 den Arbeitskreis „Geschäftsprozessmanagement mit ereignisgesteuerten Prozessketten (WI-EPK)“ der Gesellschaft für Informatik innerhalb des Fachbereichs Wirtschaftsinformatik (FB-WI). Eine umfangreiche Literaturliste zur EPK ist unter der URL http://epk.et-inf.fho-emden.de/literatur.php zugänglich.

3. Graphentheoretische Charakterisierung und Terminologie der EPK

Graphentheoretische Grundlagen

Ein **Graph** besteht in der Graphentheorie aus Knoten und Kanten, wobei jede Kante (z.B. Straße) genau zwei Knoten (z.B. Städte) verbindet und je zwei Knoten durch keine, eine oder mehrere Kanten verbunden sein können. Ein Graph wird **gerichtet** genannt, wenn die Menge der Knotenpaare (also der Kanten) eine Menge von geordneten Paaren ist. In gerichteten Graphen werden die Kanten durch Pfeile statt durch Linien dargestellt, wobei der Pfeil vom Anfangs- zum Endknoten zeigt. Ein Graph heißt **zusammenhängend**, wenn man von jedem Knoten zu jedem anderen Knoten über eine Folge von Kanten gelangt, d.h. der Graph nicht in mehrere Teile „zerfällt“.

EPK im Kontext der Graphentheorie

In dieser graphentheoretischen Terminologie ist ein EPK-Modell ein gerichteter und zusammenhängender Graph, dessen Knoten Ereignisse, Funktionen und Verknüpfungsoperatoren sind. Diese Charakterisierung entspricht der Interpretation von Rump (1999), in der neben Ereignissen und Funktionen auch Verknüpfungsoperatoren als Knoten eines EPK-Modells interpretiert werden. In der Literatur werden EPK-Modelle gelegentlich auch als bipartite Graphen eingestuft (vgl. Schütte 1998, S. 100), wobei ein Graph bipartit ist, falls es eine Partition der Knotenmenge in zwei Mengen gibt, sodass jede Kante zwischen einem Knoten der ersten Menge und einem Knoten der zweiten Menge verläuft. Eine Partition einer gegebenen Menge ist eine Menge paarweise disjunkter, nichtleerer Teilmengen der gegebenen Menge, deren Vereinigung gleich der ursprünglich gegebenen Menge ist. Diese Einstufung beschränkt folglich die Knotenmenge eines EPK-Modells auf zwei Mengen — eine Ereignis- und eine Funktionenmenge.

EPK-Terminologie

Aus Gründen der sprachlichen Klarheit wird nachfolgend die Modellierungssprache **ereignisgesteuerte Prozesskette** „EPK“ genannt, während ein unter Verwendung dieser Sprache expliziertes Modell als **EPK-Modell** bezeichnet wird. Analog kann dann auch von einem **EPK-Informationsmodell** oder einem **EPK-Referenzmodell** gesprochen werden. Gelegentlich findet sich in der Literatur statt EPK-Modell auch die Bezeichnung EPK-Schema (vgl. Nüttgens/Rump 2002, S. 65 ff.).

Frage 2: Wie können EPK-Modelle graphentheoretisch charakterisiert werden?

4. Grundlegende Sprachkonstrukte der EPK

Abb. 1 zeigt ein EPK-Modell der Kundenauftragsbearbeitung. Es beschreibt den Ablauf zur Definition und Durchführung von Prüffunktionen eines Kundenauftrags. Jedes der „Negativergebnisse“ führt zur unmittelbaren Ablehnung des Kundenauftrags, unabhängig von den Prüfergebnissen der anderen Funktionen. Die grundlegenden Sprachkonstrukte der EPK sowie ihre Repräsentationsformen werden anhand dieses Beispiels erläutert.

EPK-Sprachkonstrukte

Die Grundelemente der Modellierungssprache EPK sind Ereignisse, Funktionen, Kontrollflusskanten und Verknüpfungsoperatoren. **Ereignisse** sind die passiven Elemente der EPK. Sie beschreiben Zustände und werden durch Sechsecke dargestellt. **Funktionen**, die durch an den Ecken abgerundete Rechtecke repräsentiert werden, sind die aktiven Elemente der EPK. Der Funktionsbegriff wird in der EPK mit dem der Aufgabe

gleichgesetzt (vgl. Keller/Nüttgens/Scheer 1992, S. 8). Im Gegensatz zu einer Funktion, die ein zeitverbrauchendes Geschehen ist, ist ein Ereignis auf einen Zeitpunkt bezogen.

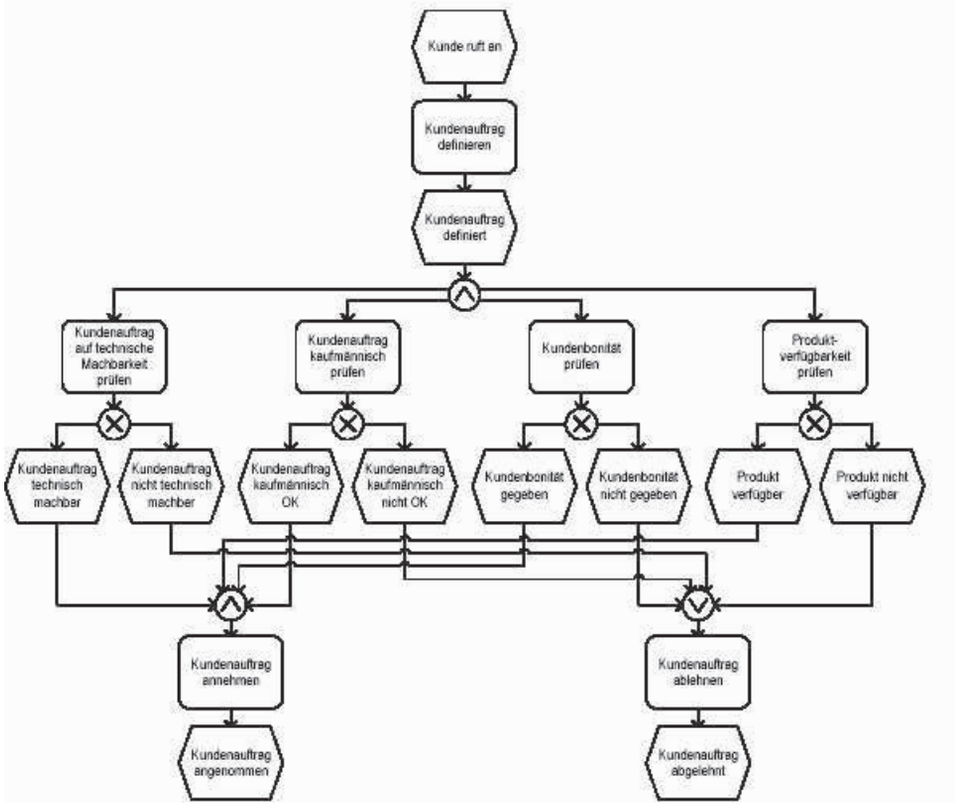


Abb. 1: EPK-Modell einer Kundenauftragsbearbeitung

Namenskonventionen

Während zur Bezeichnung der Funktionen in der Literatur (vgl. z.B. Hoffmann/Kirsch/Scheer 1992, S. 5) vorgeschlagen wird, das jeweilige Objekt der Bearbeitung und ein Verb im Infinitiv zur Kennzeichnung der zu verrichtenden Tätigkeit zu verwenden (z.B. Kundenauftrag definieren, vgl. Abb. 1), wird für Ereignisse empfohlen, das Objekt, das eine Zustandsänderung erfährt, mit einem Verb im Partizip Perfekt zu verbinden, das die Art der Änderung beschreibt (z.B. Kundenauftrag (ist) definiert, vgl. Abb. 1). Bezeichnungen und Namen von Sprachkonstrukten in Prozessmodellen werden bei Erläuterungen im Text durch Anführungsstriche hervorgehoben.

Kontrollfluss

Ereignisse lösen Funktionen aus und sind deren Ergebnis. Diese beiden Beziehungen zwischen Funktionen und Ereignissen werden durch **Kontrollflusskanten**, die durch Pfeile repräsentiert werden, dargestellt. Um auszudrücken, dass eine Funktion durch ein oder mehrere Ereignisse gestartet werden bzw. eine Funktion ein oder mehrere Ereignisse als Ergebnis erzeugen kann, werden **Verknüpfungsoperatoren** (Konnektoren) eingeführt. Dabei wird in Anlehnung an die Terminologie der Aussagenlogik zwischen **konjunktiven** „∧“, **adjunktiven** „∨“ und **disjunktiven** Verknüpfungen „⊗“ unterschieden (vgl. Abb. 1). Die entsprechenden Konnektoren werden vereinfacht als **AND-**, **OR-** bzw. **XOR-Operatoren** bezeichnet.

Syntaxregeln

Nüttgens/Rump (2002, S. 67 ff.) definieren eine EPK-Syntax, in welcher der Kontrollfluss betrachtet wird und als Knoten Ereignisse, Funktionen, Verknüpfungsoperatoren und Prozesswegweiser, die in Abschnitt 5.4 als Erweiterung der EPK eingeführt werden, zugelassen sind. Auf Basis dieser Syntax formulieren die Autoren Regeln, die der Konstruktion syntaktisch richtiger EPK-Modelle dienen:

a) Ein EPK-Modell ist ein **gerichteter und zusammenhängender** Graph.
b) Ein EPK-Modell ist in graphentheoretischer Terminologie **einfach**, wobei „einfach“ bedeutet, dass das EPK-Modell als Graph keine Schlinge (Kante mit gleichen Anfangs- und Endknoten) und keine Mehrfachkanten (mehrere Kanten zwischen Knoten) enthält.
c) **Funktionen** besitzen genau eine eingehende und genau eine ausgehende Kontrollflusskante.
d) **Ereignisse** haben genau eine eingehende und/oder genau eine ausgehende Kontrollflusskante (d.h., falls ein Ereignis nur eine Kontrollflusskante aufweist, handelt es sich um ein Start- oder Endereignis).

e) **Prozesswegweiser** haben genau eine eingehende oder eine ausgehende Kontrollflusskante.
f) **Verknüpfungsoperatoren** haben entweder eine eingehende und mehrere ausgehende (Split-Operator) oder mehrere eingehende und eine ausgehende Kontrollflusskante (Join-Operator).
g) Es gibt keinen gerichteten Kreis im EPK-Modell, der nur aus Verknüpfungsoperatoren besteht.
h) Ereignisse sind nur mit Funktionen und Prozesswegweisern (möglicherweise über Verknüpfungsoperatoren) verbunden.
i) Funktionen und Prozesswegweiser sind nur mit Ereignissen (möglicherweise über Verknüpfungsoperatoren) verbunden.
j) Nach Ereignissen folgt kein XOR- oder OR-Split-Operator im Kontrollfluss.
k) Es gibt mindestens ein Start- und mindestens ein Endereignis.

Frage 3: Aus welchen grundlegenden Sprachkonstrukten besteht die Modellierungssprache EPK?

5. Modellierungssprachliche Erweiterungen der EPK

Erweiterte EPK

In Forschung und Praxis existieren Erweiterungen der Modellierungssprache EPK, die unter anderem darauf abzielen, den Umfang der möglichen Sprachaussagen zu vergrößern oder die Handhabbarkeit umfangreicher Modelle zu verbessern. In der Literatur wird dementsprechend in vielen Fällen von der erweiterten EPK (kurz: eEPK) gesprochen (vgl. z.B. Rump 1999, S. 51). Dieser Sprachgebrauch wird hier jedoch nicht empfohlen, da keine einheitliche Meinung darüber existiert, welche Sprachkonstrukte zu einer Grundform und welche zu einer erweiterten Form der EPK gehören. Auch in der Modellierungspraxis wird diese Bezeichnung uneinheitlich verwendet.

5.1. ARIS-Sprachkonstrukte

Erweiterung der Sprachaussagen

Aus der Ableitung der EPK als zentrale Modellierungssprache der Architektur integrierter Informationssysteme (ARIS, vgl. Scheer 2001; 2002) resultieren erweiterte Aussagen, die auf dem ARIS-Sichtenkonzept aufbauen.

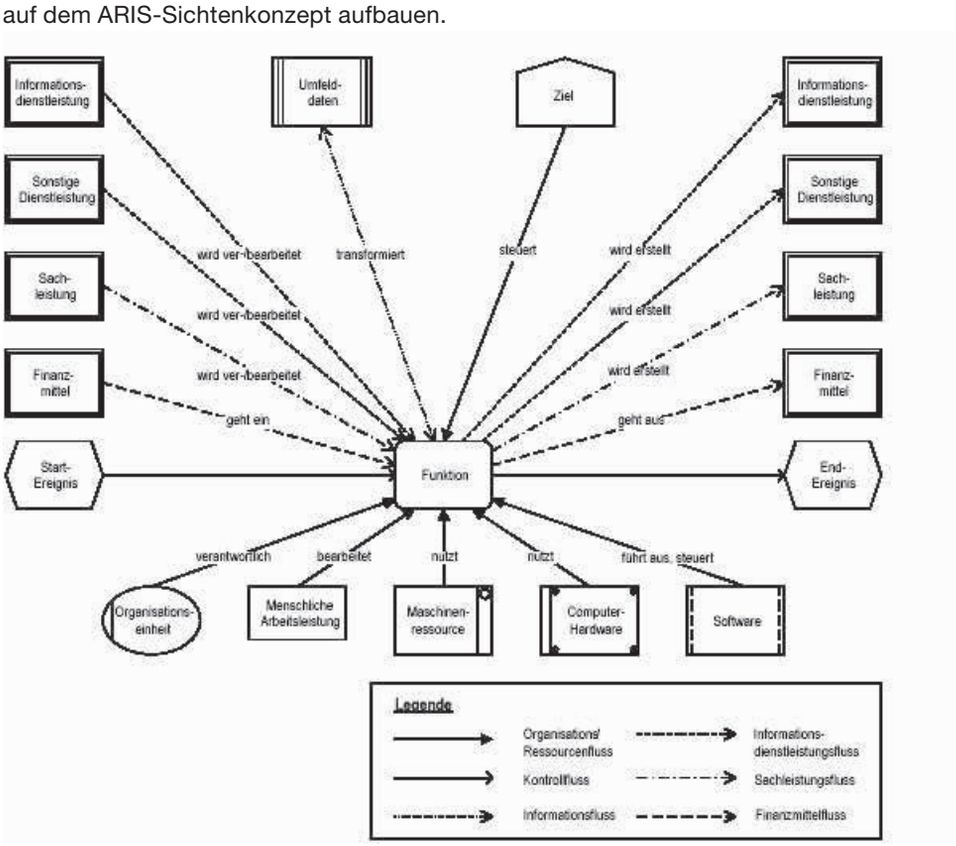


Abb. 2: Erweiterung der EPK um ARIS-Sprachkonstrukte (Quelle: Scheer 2002, S. 31)

Diese werden durch **Annotation von zusätzlichen Sprachkonstrukten** an EPK-Funktionen getroffen. So werden beispielsweise Sprachkonstrukte vorgeschlagen, die Umfeld-

daten, Nachrichten, menschliche Arbeitsleistung, maschinelle Ressourcen und Computer-Hardware, Anwendungssoftware, Leistungen in Form von Sach-, Dienst- und Informationsdienstleistungen, Finanzmittel, Organisationseinheiten oder Unternehmensziele repräsentieren (vgl. Abb. 2). Die Verbindung der Konstrukte, die nur mit Funktionen der EPK erfolgen kann, wird über Kanten hergestellt, die neben dem bereits eingeführten Kontrollfluss in Organisations-/Ressourcen-, Informations-, Informationsdienstleistungs- und Sachleistungs- sowie Finanzmittelfluss unterschieden werden (vgl. Scheer 2002, S. 31).

5.2. Geteilte Operatoren

Ein- und Ausgangsverknüpfungen

Die Verknüpfungsoperatoren können danach unterschieden werden, ob sie eine eingehende und mehrere ausgehende (Split-Operator) oder mehrere eingehende und eine ausgehende Kontrollflusskante (Join-Operator) besitzen. Eine Erweiterung des in der EPK verfolgten Operatorenkonzepts besteht darin, diese Unterscheidung durch „geteilte“ Operatoren, die sowohl eine Eingangs- als auch eine Ausgangsverknüpfung unterscheiden, aufzulösen (vgl. Scheer 1997, S. 49). Im oberen Bereich des Operators wird das logische Zeichen der Eingangs- und im unteren Bereich das der Ausgangsverknüpfung eingetragen. Sofern nur ein Ein- bzw. Ausgang auftritt, entfällt ein entsprechendes logisches Symbol. Falls genau eine Eingangs- und eine Ausgangsverknüpfung bestehen, entfällt der Knoten. Mögliche EPK-Modelle und entsprechende Interpretationen der repräsentierten Ablaufstrukturen sind in Abb. 3 dargestellt.

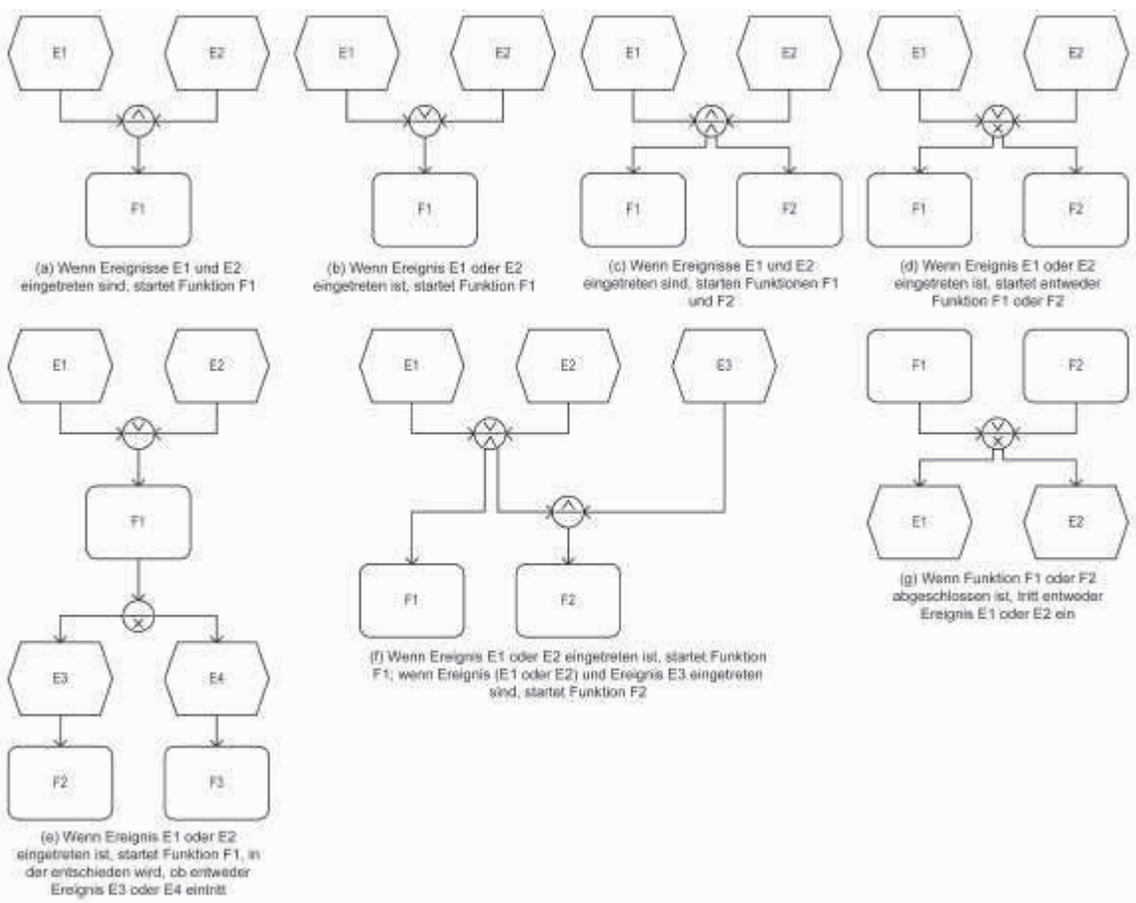

E1
E2
F1
(a) Wenn Ereignisse E1 und E2 eingetreten sind, startet Funktion F1
(b) Wenn Ereignis E1 oder E2 eingetreten ist, startet Funktion F1
F2
(c) Wenn Ereignisse E1 und E2 eingetreten sind, starten Funktionen F1 und F2
(d) Wenn Ereignis E1 oder E2 eingetreten ist, startet entweder Funktion F1 oder F2
E3
E4
F3
(e) Wenn Ereignis E1 oder E2 eingetreten ist, startet Funktion F1, in der entschieden wird, ob entweder Ereignis E3 oder E4 eintritt
(f) Wenn Ereignis E1 oder E2 eingetreten ist, startet Funktion F1; wenn Ereignis (E1 oder E2) und Ereignis E3 eingetreten sind, startet Funktion F2
(g) Wenn Funktion F1 oder F2 abgeschlossen ist, tritt entweder Ereignis E1 oder E2 ein

Abb. 3: EPK-Modelle unter Berücksichtigung geteilter Operatoren (in Anlehnung an Scheer 1997, S. 50 ff.)

Frage 4: Welcher Unterschied besteht zwischen Split- und Join-Operatoren?

5.3. Modellierung sequenzieller Abläufe

Sequenz-Operator

Des Weiteren finden sich in der Literatur Verknüpfungsoperatoren, welche auf die Verbesserung der Handhabbarkeit umfangreicher Modelle abzielen, z.B. der Sequenz (SEQ)-Operator (vgl. Priemer 1995, S. 269 ff.) oder der Entscheidungstabellen (ET)-Operator (vgl. Rosemann 1996, S. 140 ff.). Exemplarisch wird hier der SEQ-Operator vorgestellt.

Die Grundidee des SEQ-Operators liegt in der kompakten Darstellung wahlfreier Sequenzen. Beispielsweise ist denkbar, dass an einem Rohteil in der industriellen Fertigung die Arbeitsgänge „Bohren“, „Fräsen“ und „Schweißen“ in beliebiger Reihenfolge durchgeführt werden können, das Ergebnis jeder der Funktionen durch keines der beiden anderen beeinflusst wird und eine simultane Durchführung von zwei oder gar drei der Arbeitsgänge an dem Rohteil nicht möglich ist. Würde die bestehende EPK-Syntax verwendet werden, so müssten, um die Flexibilität bei der Wahl der Reihenfolge der Arbeitsgänge durch das EPK-Modell zu repräsentieren, alle sechs Möglichkeiten, die sich für die Reihenfolge der Arbeitsgänge ergeben, modelliert werden. Dies ist durch Verwendung einer Prüffunktion zweier XOR-Operatoren in Abb. 4 links dargestellt.

Beispiel

Eine wesentlich kompaktere Darstellung ergibt sich durch die Verwendung des SEQ-Operators in Abb. 4 rechts. Die Darstellung ist so zu interpretieren, dass alle Funktionen die auf den SEQ-Operator folgen, genau einmal und nacheinander auszuführen sind. Dies sind in dem Beispiel die Funktionen „Teil bohren“, „Teil fräsen“ und „Teil schweißen“, die Reihenfolge ihrer Bearbeitung spielt jedoch keine Rolle. Eine Funktion, die dem SEQ-Operator vorgeschaltet ist, prüft, welche der sechs möglichen Reihenfolgen auszuwählen ist. Um zu gewährleisten, dass das Ende der wahlfreien Sequenz eindeutig zu erkennen ist, werden die durch den SEQ-Operator getrennten Prozess-Stränge wieder durch diesen zusammengeführt.

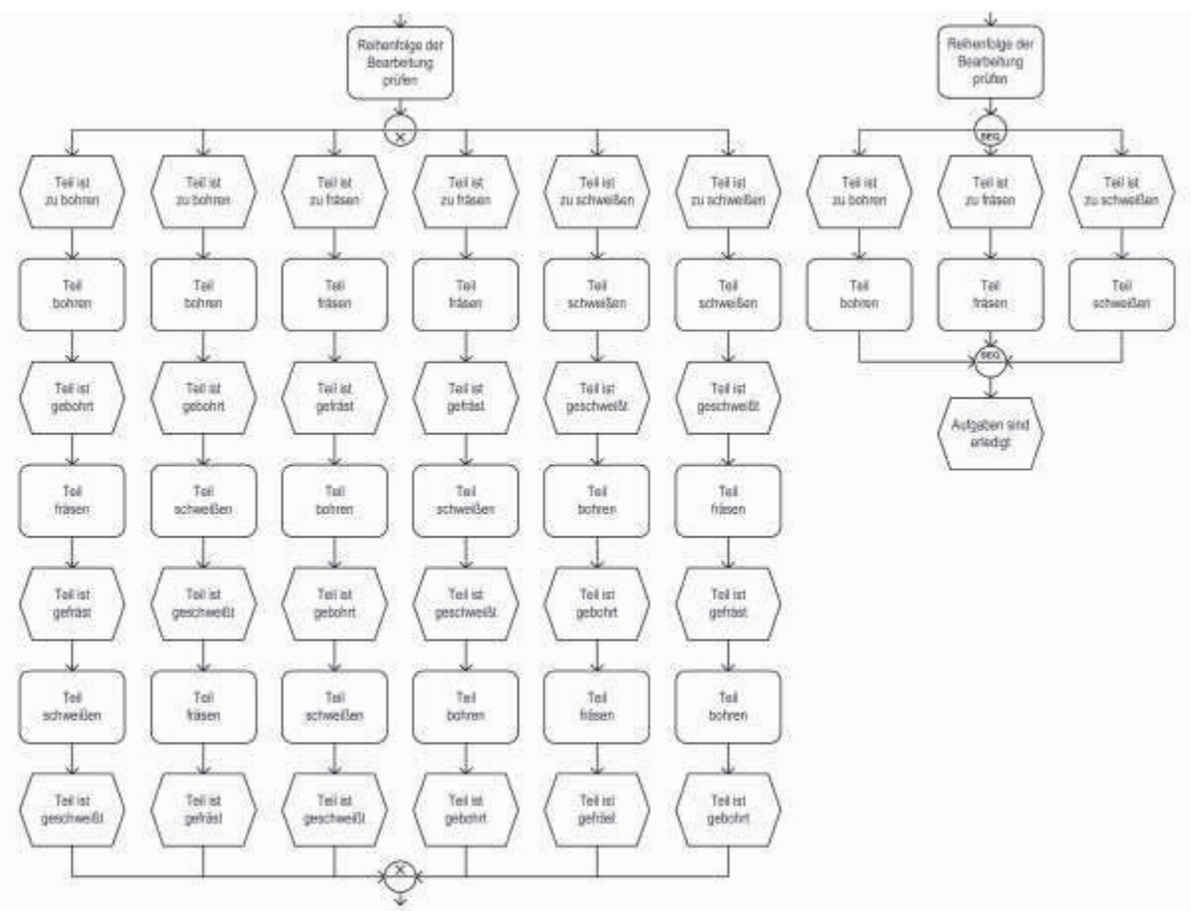


Abb. 4: Erweiterung der EPK-Syntax um den Sequenz-Operator

5.4. Prozesswegweiser und Funktionsverfeinerung

Handhabung umfangreicher EPK-Modelle

Ablauflogische Aspekte von Informationssystemen werden in der Praxis aus Gründen der Übersichtlichkeit gemeinhin nicht durch ein einziges EPK-Modell repräsentiert, sondern durch mehrere miteinander verbundene und dekomponierte Teilmodelle. Mehrere EPK-Modelle werden durch **Prozesswegweiser** miteinander verknüpft. **Funktionsverfeinerungen**, in der Modellierungspraxis häufig als **Hinterlegungen** bezeichnet, dienen der Hierarchisierung von EPK-Modellen. In Forschung und Praxis sind unterschiedliche Notationsformen für diese Erweiterungen zu finden. Abb. 5 enthält einen Notationsvorschlag, bei dem die EPK-Teilmodelle durch einen Rahmen und einen Namen gekennzeichnet sind. Bei der Dekomposition einer Funktion wird deren Name an das entsprechende Detailmodell „vererbt“. In Abb. 5 ist eine solche Dekomposition für die Funktion

F_2 vollzogen, wobei diese lediglich darin besteht, zusätzliche Sprachkonstrukte an diese EPK-Funktion zu annotieren, um die Aussage um organisatorische und leistungsbezogene Aspekte zu erweitern. Der in Abb. 5 verwendete Prozesswegweiser verknüpft die beiden EPK-Teilmodelle EPK_1 und EPK_2. Die Bezeichnung des Prozesswegweisers ist jeweils so gewählt, dass dieser namentlich auf das entsprechende Modell verweist. So zeigt beispielsweise der Prozesswegweiser EPK_2 im EPK-Modell EPK_1 auf das EPK-Modell EPK_2.

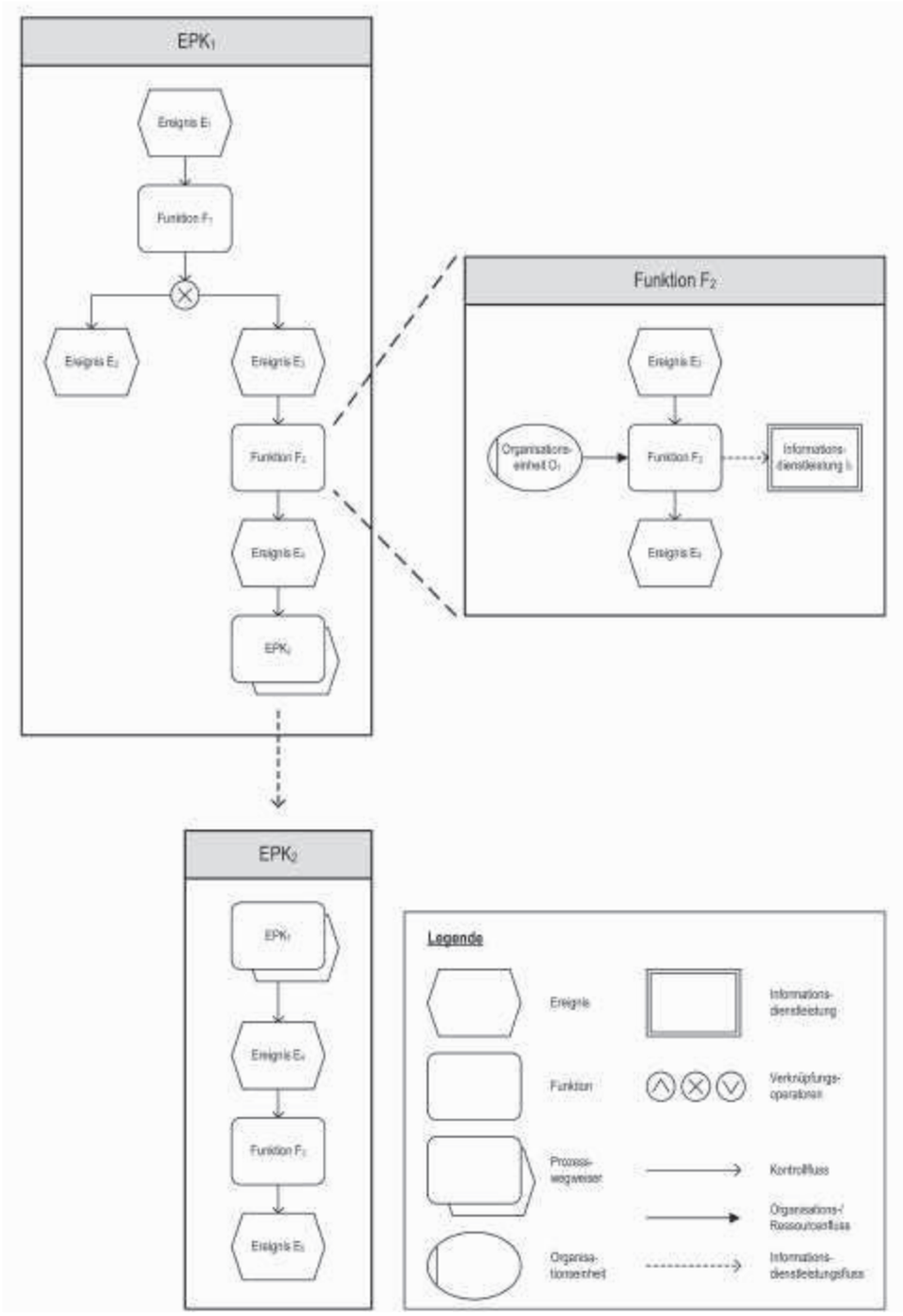


Abb. 5: Erweiterung der EPK um Prozesswegweiser und Funktionsverfeinerung

Navigationsunterstützung

Wie bedeutungsvoll insbesondere das Konstrukt der Funktionsverfeinerung in der Modellierungspraxis ist, lässt sich anhand des in Abb. 6 dargestellten Ausschnitts des SAP R/3-Referenzmodells veranschaulichen. Da dieses Modell aus mehr als 1.000 Geschäftsprozessen besteht, wird eine überblicksartige grafische Repräsentation erschwert. Durch die Verwendung der Funktionsverfeinerung entsteht eine Hierarchisierung der Teilprozesse. Computergestützte Modellierungswerkzeuge, wie das ARIS-Toolset, nutzen diese Hierarchisierung, um den Benutzern eine Navigation der Prozessmodelle zu ermöglichen.

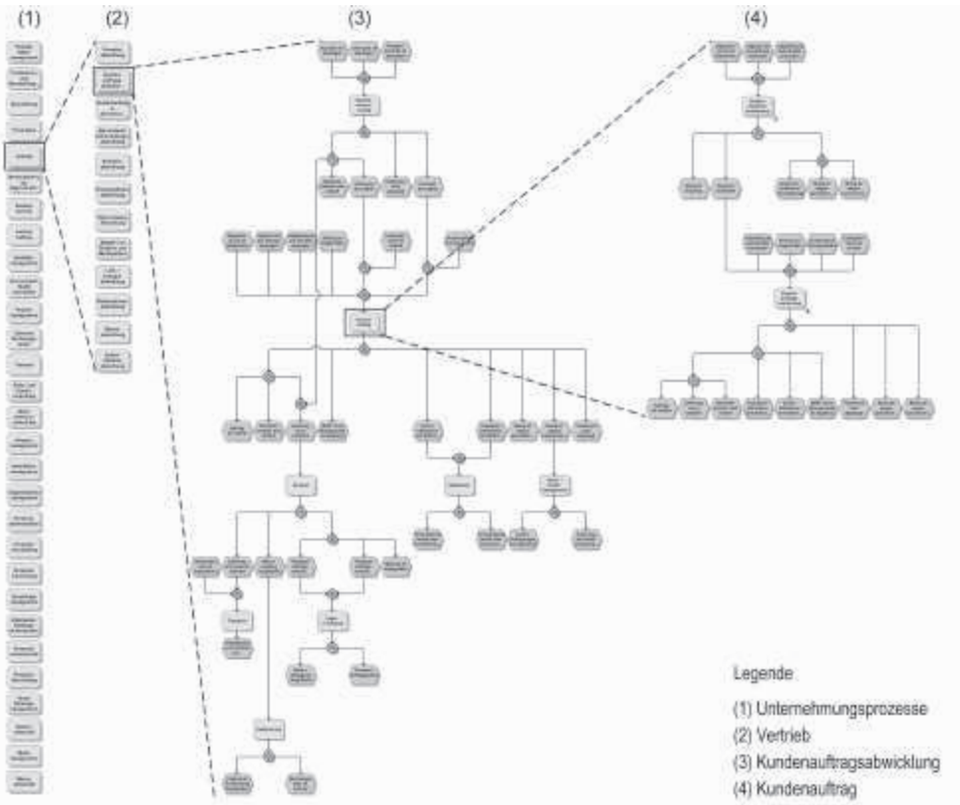


Abb. 6: SAP R/3-Referenzmodell (Ausschnitt, Quelle: IDS Scheer AG 2003)

Frage 5: Welche Bedeutung hat das EPK-Konstrukt der Funktionsverfeinerung?

5.5. Variantenmanagement

EPK-Referenzmodelle und Varianten

Eine andere Erweiterung der EPK resultiert aus dem Arbeitsgebiet der Referenzmodellierung (vgl. Becker et al. 2002). Versteht man Referenzmodelle generell als Informationsmodelle zur Unterstützung der Konstruktion anderer Modelle, dann besteht die bei der Anwendung eines Referenzmodells von einem Modellierer zu vollziehende Aufgabe in der **Referenzmodelladaption**. Die mit diesem Begriff charakterisierte Ableitung spezifischer Modelle aus einem Referenzmodell entspricht im übertragenen Sinne der Bildung von Varianten des Referenzmodells (vgl. Schütte 1998, S. 207 ff.). So könnten beispielsweise aus einem Referenzmodell „Fertigung“ die unternehmensspezifischen Modelle „Informationsmodell stückorientierte Fertigung Unternehmen U_1“ oder „Informationsmodell prozessorientierte Fertigung Unternehmen U_2“ als Varianten abgeleitet sein. Um eine solche Ableitung zu gewährleisten, müssen alternative Bausteine in den Referenzmodellen vorgehalten werden, die der Modellnutzer während der Adaption beispielsweise unter Angabe unternehmensspezifischer Gegebenheiten beibehält, ergänzt oder entfernt.

Buildtime- und Runtime-Modelle

Diese Berücksichtigung von Alternativen in den Referenzmodellen geht jedoch zu Lasten der „Lauffähigkeit“ der Modelle und impliziert die für die Referenzmodellierung charakteristische Unterscheidung von Informationsmodellen auf **Buildtime- und Runtime**-Ebene. Während Modelle auf Buildtime-Ebene nicht ablauffähig sein müssen, enthalten Modelle auf Runtime-Ebene nur ablauffähige Konstrukte (vgl. Schütte 1998, S. 244, Fn. 154).

Operatoren zur Referenzmodelladaption

Während die Berücksichtigung von ablauflogischen Alternativen auf Ebene der Runtime bereits durch die Verknüpfungsoperatoren OR und XOR gewährleistet ist, muss die EPK zur Darstellung von Alternativen auf Buildtime-Ebene um Konstrukte erweitert werden. Zur Darstellung dieser Alternativen werden so genannte **Buildtime-Operatoren** eingeführt (vgl. Schütte 1998, S. 244 ff.). Zur Veranschaulichung dieser Operatoren greift Abb. 7 das EPK-Modell der Kundenauftragsbearbeitung aus Abb. 1 erneut auf.

Während die Funktionen zur Prüfung des Kundenauftrags auf zwei reduziert sind, ist das Modell um die Verwendung des Buildtime-Operators XOR_B ergänzt (vgl. Schütte 1998, S. 246). Zur Verdeutlichung, dass dieser Operator nur auf Buildtime-Ebene agiert, ist er im Gegensatz zu den Runtime-Operatoren zweifach umrandet.

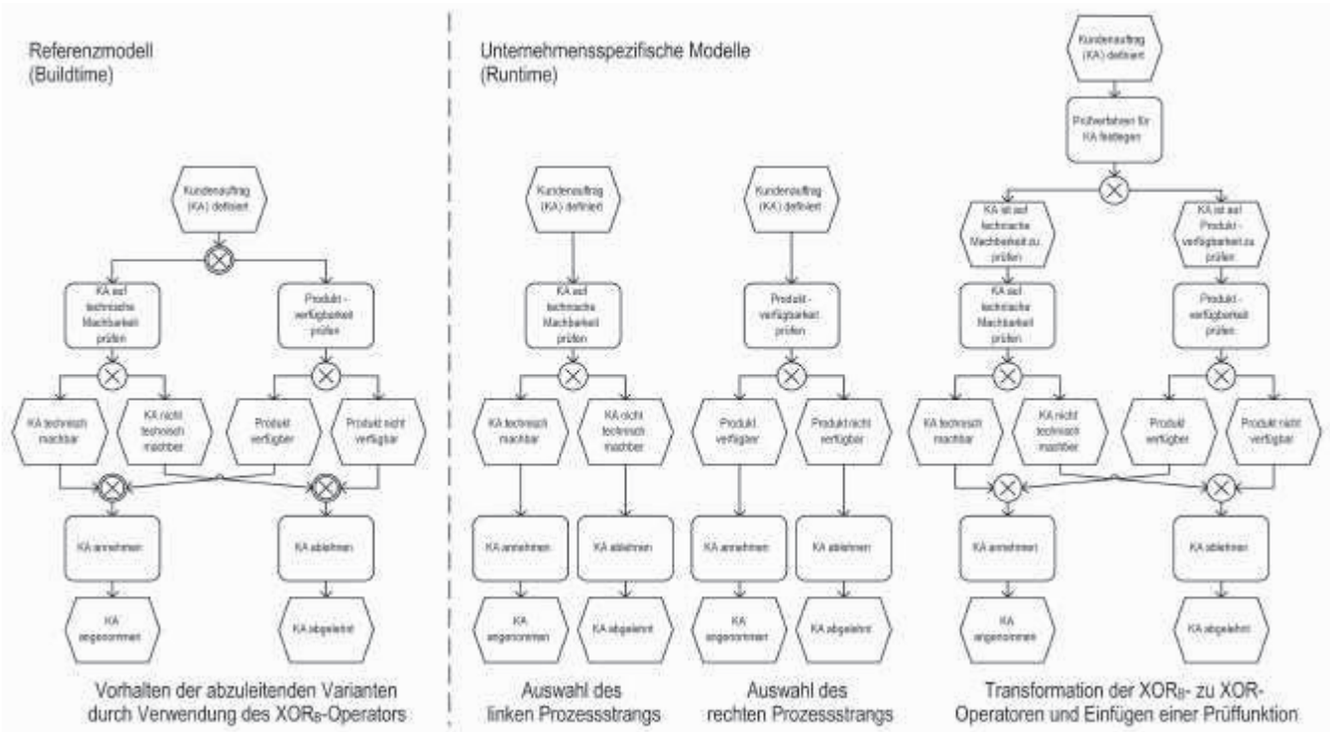


Abb. 7: Ableitung unternehmensspezifischer EPK-Modelle aus einem EPK-Referenzmodell mithilfe von Buildtime-Operatoren

Auflösung von Buildtime-Operatoren

Bei der Ableitung unternehmensspezifischer Modelle aus dem Referenzmodell ergeben sich drei mögliche Varianten: Entweder wird die Disjunktion auf Buildtime-Ebene zu einer Disjunktion auf Ebene der Runtime oder sie führt zur Auswahl eines der beiden Prozessstränge (vgl. Abb. 7). Neben dem disjunktiven Buildtime-Operator XOR$_B$ existieren weitere Operatoren, mit deren Hilfe Varianten aus Referenzmodellen abgeleitet werden können (vgl. Schütte 1998, S. 251). Für jeden dieser Buildtime-Operatoren ist festzuhalten, welche Runtime-Operatoren oder Prozess-Stränge sich aus ihm deduzieren lassen.

Frage 6: Welcher Sachverhalt begründet die Unterscheidung von EPK-Modellen auf Buildtime- und Runtime-Ebene?

6. Fazit und Ausblick

Die ereignisgesteuerte Prozesskette ist eine in Wissenschaft und Praxis weit verbreitete Modellierungssprache zur Repräsentation von Geschäftsprozessen. Sie ist darüber hinaus Gegenstand aktueller Forschungen. Diese Arbeiten widmen sich einerseits der Konstruktion von Prozessmodellen für bestimmte Anwendungsdomänen, womit insbesondere auf den Mangel branchenspezifischer Ausrichtungen von Referenzmodellen reagiert wird. Andererseits werden modellierungssprachliche und methodische Aspekte der Konstruktion von EPK-Modellen untersucht. Hierbei werden vor allem die Möglichkeiten einer formalen Spezifikation der EPK-Syntax und Semantik geprüft sowie der Bezug zu verwandten Modellierungsansätzen wie Petri-Netzen und UML-Objektmodellen analysiert. Zur Lektüre entsprechender Arbeiten wird auf die umfangreiche Literaturliste der EPK-Community verwiesen (vgl. http://epk.et-inf.fho-emden.de/literatur.php).

Literaturempfehlungen:

Becker, J./Algermissen, L./Delfmann, P./Knackstedt, R.: Referenzmodellierung. In: WISU, 31. Jg., (2002), S. 1392 - 1395.

Hoffmann, W./Kirsch, J./Scheer, A.-W.: Modellierung mit Ereignisgesteuerten Prozeßketten, Methodenhandbuch. In: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Nr. 101. Saarbrücken Dezember 1992.

IDS Scheer AG (Hrsg.): ARIS Toolset, ARIS Version 6.2.1.31203. Saarbrücken 2003.

Keller, G./Nüttgens, M./Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“. In: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Nr. 89. Saarbrücken 1992.

Keller, G./Teufel, T.: SAP R/3 prozeßorientiert anwenden. Iteratives Prozess-Prototyping mit Ereignisgesteuerten Prozessketten und Knowledge Maps. 3. Aufl., Bonn et al. 1999.

Nüttgens, M./Rump, F.J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: Desel, J./Weske, M. (Hrsg.): Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen (Promise 2002). Hasso-Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam, 9.-11. Oktober 2002. Bonn 2002, S. 64 - 77.

Priemer, J. (1995): Entscheidungen über die Einsetzbarkeit von Software anhand formaler Modelle. Sinzheim 1995.

Rosemann, M.: Komplexitätsmanagement in Prozessmodellen. Methodenspezifische Gestaltungsempfehlungen für die Informationsmodellierung. Wiesbaden 1996.

Rump, F.J.: Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten: Formalisierung, Analyse und Ausführung von EPKs. Stuttgart et al. 1999.

Scheer, A.-W.: ARIS — Vom Geschäftsprozess zum Anwendungssystem. 4. Aufl., Berlin et al. 2002.

Scheer, A.-W.: ARIS — Modellierungsmethoden, Metamodelle, Anwendungen. 4. Aufl., Berlin et al. 2001.

Scheer, A.-W.: Wirtschaftsinformatik: Referenzmodelle für industrielle Geschäftsprozesse. 7. Aufl., Berlin et al. 1997.

Schütte, R.: Grundsätze ordnungsmäßiger Referenzmodellierung: Konstruktion konfigurations- und anpassungsorientierter Modelle. Wiesbaden 1998.

Die Fragen werden im WISU-Repetitorium beantwortet.

Die Fallstudie aus der Wirtschaftsinformatik

IT-Outsourcing

Prof. Dr. Andreas Gadatsch, St. Augustin

1. Fallbeschreibung

Untersucht wird ein Industriekonzern, dessen Produkte je nach Witterung nachgefragt werden. Der hohen Nachfrage während der kälteren Monate folgt ein Nachfragerückgang während der wärmeren Perioden. Das Unternehmen ist weltweit vertreten. Die IT-Kapazitäten des Unternehmens lassen sich nicht an die **schwankende Nachfrage** anpassen. Zu geringe Kapazitäten in der Hauptsaison stehen nicht ausgelasteten Kapazitäten in der Nebensaison gegenüber. Die langen Antwortzeiten in der Hauptsaison führen zu regelmäßig wiederkehrenden Beschwerden der Anwender.

Die anfallenden IT-Kosten werden nach einem Schlüssel auf alle Abteilungen des Hauses verteilt. Das Abrechnungsverfahren führt dazu, dass kostenbewusstes Verhalten der Anwender nicht belohnt wird. Als Konsequenz werden die Ist-Verrechnungen von den Anwendern ignoriert. Ein IT-Anwender, der relativ hohe IT-Kosten durch zahlreiche Auswertungs- und Sonderprogramme verursacht, bezeichnet die Belastung mit IT-Kosten auf seinem Kostenstellen-Betriebsabrechnungsbogen als „Spielgeld“, auf das er keinen Einfluss habe.

Der Konzern setzt eine mittlerweile veraltete großrechnergestützte ERP-Software ein, deren **Migration auf das modernere Nachfolgeprodukt** nicht mehr lange hinauszuzögern ist. Das liegt insbesondere am ständigen Personalmangel im Rechenzentrum und an der fehlenden Funktionalität der Anwendungssoftware. Der im Rahmen einer Machbarkeitsstudie vorgesehene Release-Wechsel soll sukzessiv erfolgen (zunächst Umstellung des Rechnungswesens, danach der Logistik etc.), da ein Big-Bang die Betriebsbereitschaft des Unternehmens gefährden würde. Die gleichzeitige Betreuung von Neu- und Altsystem wird die IT-Ressourcen stark belasten. Eine vorübergehende Erweiterung des Rechenzentrums (Personal, Rechner, Büros etc.) ist bei dieser Strategie unausweichlich.

Die **Personalressourcen** der Anwendungsentwicklung und -betreuung sind weitgehend erschöpft. Das Personal ist überwiegend mit Wartungsarbeiten des ERP-Systems und weiterer Individualsoftware beschäftigt. Die wenigen Neuentwicklungen betreffen vor allem Internet-Anwendungen. Viele Projekte werden nicht termingerecht fertig, weshalb zahlreiche Mitarbeiter bei mehreren Projekten mitarbeiten. Für viele Aufgaben werden externe Mitarbeiter eingesetzt. Die Fachabteilungen haben zahlreiche Eigenentwicklungen mithilfe von PC-Tools für zum Teil geschäftskritische Applikationen erstellt. An Neuentwicklungen oder die Einführung des neuen ERP-Systems ist mit den vorhandenen Personalressourcen nicht zu denken.

Wegen des Kostendrucks wurden IT-Schulungen aufgeschoben. Anwenderfehler, die auf mangelnde Schulung zurückzuführen sind, häufen sich. Die Behebung dieser Fehler bindet weitere Kapazitäten in der IT-Abteilung. Der hohe Wartungsanteil und die fehlende Weiterbildung haben zu hoher **Unzufriedenheit** bei den IT-Mitarbeitern und bei Schlüsselmitarbeitern in den Fachabteilungen geführt. Insbesondere Periodenabschlüsse führen bei vielen Mitarbeitern zur Nacht- und Wochenendarbeit. Immer mehr wichtige Mitarbeiter kündigen. Die Lücken müssen zu hohen Kosten durch externe Berater geschlossen werden.

2. Aufgaben

Der Vorstand erwägt die Auslagerung der IT an einen spezialisierten IT-Dienstleister und beauftragt den Chief Information Officer (CIO), die Situation zu analysieren und Verbesserungsvorschläge zu erarbeiten. Er wünscht einen Anforderungskatalog für die Ausschreibung und einen Vorschlag für die Aufgabenteilung, aus der hervorgeht, welche Aufgaben beim Unternehmen bleiben und welche der IT-Dienstleister übernehmen soll.

3. Lösung

Der CIO überzeugt den Vorstand davon, dass sich die Probleme des Unternehmens durch Outsourcing aller IT-Aktivitäten lösen lassen. Die von ihm dargelegten wichtigsten **Gründe für die Auslagerung** der IT-Aktivitäten: